

Asymmetric Homomorphic Encryption Transformation for Securing Distributed Data Storage in Wireless Sensor Networks

Diploma Thesis at Integrated Circuits and Systems Laboratory
Department of Computer Science
Technical University of Darmstadt
and
NEC Europe Network Laboratories

from
cand. dipl.-inform.
Osman Ugus
(ugus@netlab.nec.de)

Supervisor:
Prof. Dr.-Ing. Sorin A. Huss
Dipl.-Inform. Ralf Laue
Dr. habil. Dirk Westhoff (NEC Europe Ltd.)

April 2007

Bu alıřmayı annem Meryem ve babam Ismail'e adıyorum
Dedicated to my parents Meryem and Ismail

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt,

Acknowledgments

First of all, I would like to thank Prof. Dr. Sorin A. Huss for giving me the opportunity to do this thesis at NEC Europa Ltd. in Heidelberg. I also would like to thank Dr. Dirk Westhoff who helped me with solving the problems, which I faced during my internship. Further, I would like to thank Ralf Laue for his worthwhile suggestions and helpful discussions. I would like to thank NEC Europa Ltd. for financial support.

Finally, and most importantly I want to thank my girl friend Yue Chang for encouragement and emotional support.

Abstract

In asynchronous wireless sensor networks it is not possible to transmit the monitored data to an authorized recipient in real-time. Thus, the data needs to be stored within the network in distributed fashion. However, due to the resource constraints of the sensor nodes the amount of data to be stored and transmitted must be minimized. One popular technique for this is in-network data aggregation. But because wireless sensor networks are employed in public environments, the data stored in the network has to be protected against the attacks. As solution for asynchronous wireless sensor networks *tiny persistent encrypted data storage* (TinyPEDS) has been proposed. TinyPEDS provides in-network aggregation of encrypted data and stores the aggregated data in a replicated and distributed manner. Since the data being aggregated is encrypted, the encryption scheme needs to support an additive homomorphic operation on ciphertexts.

This thesis studies the implementation and integration of such an encryption scheme, namely the *elliptic curve ElGamal encryption* (EC-ElGamal) in TinyPEDS. In order to keep the resource consumption at a minimum, several algorithms for operating at finite field and elliptic curve level are analyzed and compared according to their timing values and code sizes. Several trade-offs and their effects on memory usage, code size and code efficiency are discussed. Moreover, a new approach for reducing multi-precision addition over $GF(p)$ is proposed. In order to enhance the performance and decrease the unnecessary overhead in resource consumption, several techniques are employed which ensure an appropriate programming style.

Contents

Abstract	ix
1 Introduction	1
2 Wireless sensor networks	3
2.1 Problem definition	3
2.2 Network model	4
2.3 Threat models	6
2.3.1 Dolev-Yao threat model	7
2.3.2 Extended Dolev-Yao threat model	8
2.4 Concealed data aggregation	10
2.5 Encryption Schemes	11
2.5.1 Requirements for an appropriate encryption scheme	11
2.5.2 Selecting an appropriate encryption scheme	13
2.5.2.1 Symmetric encryption scheme candidates	13
2.5.2.2 Asymmetric encryption scheme candidates	15
2.6 Distributed storage of encrypted and aggregated data	17
2.7 Query flooding and query response	19
2.7.1 Controlled query flooding	19
2.7.2 Aggregated data response	22
2.8 Approaches for increasing data availability	23
2.8.1 Disaster model	23
2.8.2 Techniques for restoring the lost data	24

3	Constraint analysis	27
3.1	Implementation platform	27
3.2	Dealing with constraints by choosing an appropriate programming style	28
3.2.1	Minimizing required memory size	29
3.2.2	Minimizing code size	30
3.2.3	Increasing code efficiency	31
4	Design decisions	33
4.1	Software architecture	33
4.2	Finite fields	33
4.2.1	Binary field $GF(2^m)$	34
4.2.2	Prime field $GF(p)$	34
4.2.3	Analysis of described finite fields	35
4.3	Finite field arithmetic over $GF(p)$	35
4.3.1	Multi-precision addition	36
4.3.2	Multi-precision subtraction	36
4.3.3	Multi-precision multiplication	37
4.3.3.1	Montgomery multiplication	38
4.3.3.2	Schoolbook multiplication	38
4.3.3.3	Comba multiplication	40
4.3.3.4	Hybrid multiplication	42
4.3.3.5	Karatsuba multiplication	43
4.3.3.6	Analysis of described multiplication algorithms	45
4.3.4	Multi-precision squaring	46
4.3.5	Modular reduction	46
4.3.5.1	Montgomery reduction	47
4.3.5.2	Barrett reduction	47
4.3.5.3	Reduction methods for pseudo-mersenne primes	48
4.3.5.4	Analysis of the described modular reduction algorithms	49
4.3.5.5	A new approach for modular multi-precision addition	49
4.4	Elliptic curve arithmetic over $GF(p)$	50
4.4.1	Introduction to the elliptic curves	50
4.4.2	Elliptic curve point addition	51

4.4.3	Elliptic curve point doubling	51
4.4.4	Selecting the coordinate system	52
4.4.4.1	Affine coordinate system	52
4.4.4.2	Projective coordinate system	53
4.4.4.3	Jacobian coordinate system	54
4.4.4.4	Chudnovsky-Jacobian coordinate system	54
4.4.4.5	Modified Jacobian coordinate system	55
4.4.4.6	Mixed coordinate system	55
4.4.4.7	Analysis of described coordinate systems	56
4.4.5	Scalar point multiplication	57
4.4.5.1	Mathematical background	60
4.4.5.2	Left-to-Right binary method	60
4.4.5.3	Right-to-Left binary method	60
4.4.5.4	Analysis of the presented binary methods	61
4.4.6	Speeding up scalar point multiplication	61
4.4.6.1	Reducing the number of point doublings and additions	61
4.4.6.1.1	Interleave Method	61
4.4.6.1.2	Shamir method	62
4.4.6.1.3	Analysis of the presented methods	64
4.4.6.2	Reducing the number of point additions	65
4.4.6.2.1	Non adjacent form (NAF)	65
4.4.6.2.2	Mutual opposite form (MOF)	66
4.4.6.2.3	Analysis of the presented signed representations	67
4.4.6.2.4	Re-Analysis of the Interleave method	68
5	Implementation	71
5.1	Data representation	71
5.2	Software Components	73
5.2.1	ECElGamalM	73
5.2.2	ECCArithC	74
5.2.2.1	ECCArithM	74
5.2.2.2	FFArithM	75
5.2.3	secpXXXr1	75
5.2.4	RandomLfsrC	76

6	Implementation results and evaluation	77
6.1	Implementation results for the finite field arithmetic	77
6.1.1	Execution time and code size	77
6.1.2	Power consumption	79
6.2	Implementation results for the elliptic curve arithmetic	79
6.2.1	Execution time and code size	79
6.2.2	Power consumption	82
6.3	Implementation results for the EC-ElGamal encryption scheme . . .	83
6.3.1	Execution time and code size	83
6.3.2	Power consumption	84
6.4	Summary	84
7	Summary and Conclusions	85
	Literatur	89

List of Figures

2.1	Network model in TinyPEDS	5
2.2	Dolev-Yao threat model	7
2.3	Extended Dolev-Yao threat model	8
2.4	Data aggregation in wireless sensor networks	11
2.5	Concealed data aggregation in wireless sensor networks	11
2.6	TinyPEDS cluster structure	18
2.7	Disaster model	23
3.1	Mica-Z mote	28
4.1	Elliptic curve ElGamal design architecture	34
4.2	Graphical representation of multi-precision multiplication	42
4.3	Graphical representation of the Karatsuba multiplication	45
4.4	Graphical representation of point addition	51
4.5	Graphical representation of point doubling	52
5.1	Graphical representation of elliptic curve ElGamal implementation . .	73
5.2	Graphical representation of elliptic curve arithmetic implementation .	74

List of Tables

2.1	Comparison of public key encryption scheme candidates	15
3.1	Effects of several programming techniques on resource consumptions .	32
4.1	The memory performance of the multiplication algorithms	46
4.2	Computational efficiencies of point additions and doublings	56
4.3	Computational efficiencies of point additions in mixed coordinates. . .	57
4.4	Computational efficiencies of point doublings in mixed coordinates . .	57
4.5	Computational efficiencies of Interleave and Shamir method	64
4.6	Performance of Interleave and Shamir method in point doublings . . .	64
4.7	Table for MOF to w MOF mappings	67
4.8	Comparison of signed representation methods	68
4.9	Costs for the Interleave method for performing point multiplication .	69
6.1	Performance of the finite field arithmetic operations	78
6.2	Performance comparison with TinyECC for finite field arithmetic . .	78
6.3	Summary of the comparison with TinyECC for finite field arithmetic	79
6.4	Power consumption of the finite field arithmetic operations	79
6.5	Comparison of the scalar point multiplication implementations	80
6.6	Power consumption of the elliptic curve arithmetic operations	82
6.7	Comparison of the EC-ElGamal encryption scheme implementations .	83
6.8	Comparison of the homomorphic addition implementations	83
6.9	Power consumption of the EC-ElGamal encryption and hom. addition	84

1. Introduction

In the past years wireless sensor networks have become more and more popular in many fields of life. There exists a large number of application domains for wireless sensor networks such as environment observation (e.g. temperature, humidity, seismic activity, and traffic control), healthcare, and even military applications.

Wireless sensor networks may be divided into two subgroups, namely asynchronous and synchronous wireless sensor networks. In synchronous wireless sensor networks the monitored data is transmitted to a reader device in real-time. In asynchronous wireless networks, however, the monitored data is transmitted to a reader device only seldomly. Therefore, asynchronous wireless sensor networks need to store the data in the network in a distributed manner. Due to the following reasons the implementation of such a distributed data storage for wireless sensor networks is very challenging.

The first problem is that due to the limited storage capacity of the nodes, the storage capacity of the whole sensor network is restricted. Thus, it is necessary to reduce the amount of the data to be stored without losing information. The second problem is the limited power capacity of the nodes. The distributed storage requires data transmission between sensor nodes. Since the data transmission affects the power consumption, techniques for minimizing it are mandatory. Finally, since the nodes are in general equipped with non-tamper-resistant hardware and wireless sensor networks are usually employed in a public environment, the data in the nodes must be protected and concealed.

One popular method employed for minimizing the data transmission and data storage is *in-network data aggregation*. In this method, the monitored data is expressed in a condensed form. This means that instead of storing all data sensed by several nodes, the network needs to store only condensed values such as the sum of these values. In order to secure the data stored in the network, data encryption techniques are employed. However, the resource restrictions of the nodes make the implementation of encryption algorithms with satisfying security level and performance challenging. Furthermore, in-network data aggregation becomes problematic when the data being aggregated is encrypted. In such case the encryption scheme

needs to allow homomorphic addition of ciphertexts. As solution for these problems, Giraio *et al.* propose *tiny persistent encrypted data storage* (TinyPEDS) in [23].

In this thesis, we study and implement the additive homomorphic elliptic curve ElGamal (EC-ElGamal) encryption scheme, which is employed for securing distributed storage and transmission of aggregated data in TinyPEDS.

The rest of this thesis is organized as follows. Chapter 2 introduces the proposed TinyPEDS approach. The constraints stemming from implementation platform and their effects on the implementation are analyzed in Chapter 3. Chapter 4 studies several basic algorithms required at finite field and elliptic curve level and compares them in terms of code size, code efficiency, and memory usage in order to select the most suitable algorithms. Chapter 5 presents how the elliptic curve ElGamal encryption scheme is implemented on a target platform, i.e. Mica-Z mote. The code size, speed and memory requirement of our implementation are compared with existing implementations from literature in Chapter 6. Finally, Chapter 7 presents the conclusion and future works.

2. Wireless sensor networks

"A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion, or pollutants, at different locations." [64].

Although wireless sensor networks are originally motivated by military applications such as battlefield surveillance and counterterrorism, nowadays they are also employed for civilian applications such as healthcare applications, home automation, and traffic control [26]. Another popular application field of wireless sensor networks is the monitoring a geographical area for environmental data. With respect to traffic classes wireless sensor networks may be divided into two groups, namely *synchronous* and *asynchronous* wireless sensor networks [23].

- **Synchronous wireless sensor networks:**

Synchronous wireless sensor networks are employed for real-time monitoring applications such as traffic monitoring where the monitored data is fluctuating and transmitted to the authorized reader device in real-time.

- **Asynchronous wireless sensor networks:**

In contrast to synchronous wireless sensor networks, the data monitored in asynchronous wireless sensor networks is not fluctuating and transmitted to the authorized reader device only when requested. Therefore, wireless sensor applications with asynchronous character need to store the monitored data itself.

In the following the tiny persistent encrypted data Storage (TinyPEDS) is described. TinyPEDS, proposed by Girao, Westhoff, Mykletun, and Araki in [23], enables encrypted and aggregated data storage in asynchronous wireless sensor networks. The main reference for this chapter is [23].

2.1 Problem definition

Asynchronous wireless sensor networks monitor the environment continuously, while the monitored data is picked up by the authorized reader only seldomly. Therefore,

wireless sensor networks with asynchronous character need to provide the functionality for persistent data storage within the network. In general, sensor nodes are spread out over a geographical area being monitored. Therefore, the data storage must be distributed among sensor nodes. However, since the sensor nodes have limited power and storage capacity, the implementation of a distributed database functionality for wireless sensor networks is much more challenging than for conventional server systems. The problems are summarized in the following.

The first problem is the limited storage capacity of the nodes. This means that the storage capacity of the whole sensor network with thousands or even millions of sensor nodes is not even comparable to the storage capacity of a single conventional desktop system or even laptop. Therefore, it is necessary to reduce the amount of data being stored without losing information needed by the application. The second problem is the limited power capacity of the nodes. For a distributed data storage, not only the data transmission between sensor nodes is necessary, but also some database query and response operations are needed to grab and evaluate the data needed by the application. All of these operations need CPU cycles to be performed and, therefore, result in power consumption. The third problem is that in general, the nodes are equipped with not tamper-resistant hardware and wireless sensor networks are deployed in a public environment. Thus, the data stored on the sensor nodes must be protected and concealed. The data protection can be achieved by encryption. However, this increases resource consumption as well. The fourth problem is that some sensor nodes may exhaust earlier than the others. Therefore, data storage replication is necessary to increase the availability and reliability of the monitored data and the wireless sensor network, respectively. However, such an approach has the drawback that the resource consumption for storage, data queries and data response operations is increased. Finally, the resource consumption within the network must be equally distributed among all sensor nodes. Therefore, the implementation of load balancing protocols are required, which increases storage and power consumption as well.

All these challenges imply that the difficulties of implementing secured and distributed data storage in wireless sensor networks stem from the resource restrictions of the nodes. Since the resource consumption in the network increases linearly with the amount of data being processed, the use of techniques to minimize it is mandatory. One well-known technique is the data aggregation, which is employed to compute statistical values such as the sum, average, variance, minimum, or maximum of the sensed values.

As solution for the previously outlined problems, TinyPEDS is proposed. TinyPEDS provides an aggregated and replicated data storage, while keeping the resource consumption small. The following section describes the asynchronous network model used in TinyPEDS.

2.2 Network model

In the network model of TinyPEDS, a node exhibits one of four roles, namely *aggregator*, *sensor*, *forwarder*, and *idle* node. The aggregator node computes an aggregation function over the sensed values received from its neighbors. Data monitoring is performed by the sensor nodes. The transmission of the monitored data towards a

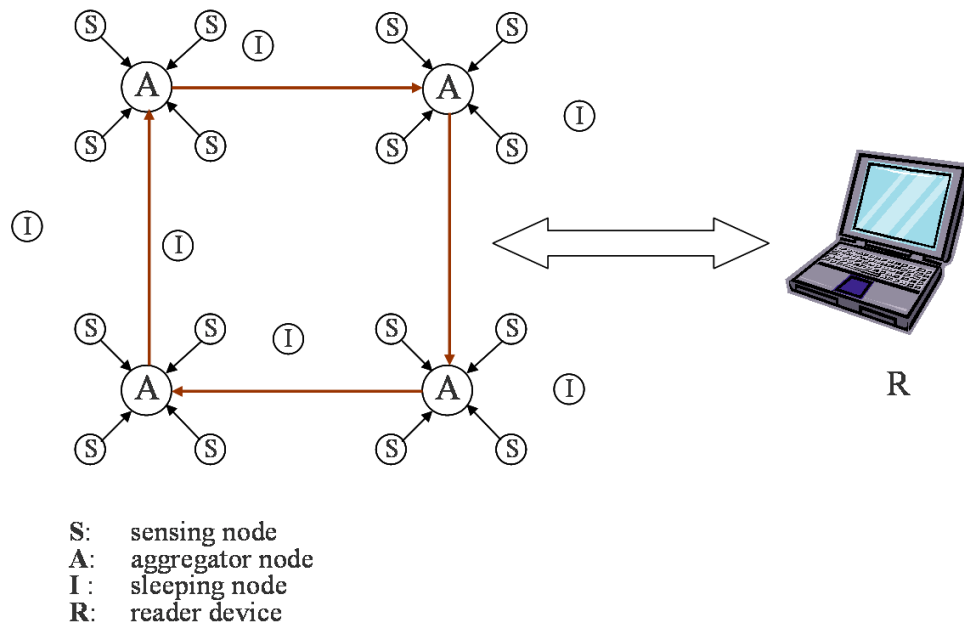


Figure 2.1: Network model in TinyPEDS

reader device or any other node is performed in hop-by-hop manner over forwarder nodes. The reader device is generally a mobile device, which can be located anywhere within the network, in order to receive the monitored data. However, in general the transmission distance to the reader device is minimum, when it is located in the center of the sensor network. An example of such a mobile device is a laptop. Therefore, the reader device is assumed to have unlimited power and storage capacity. Finally, a sensor node, which is in sleeping mode, is defined as idle node. An example for the network model employed in TinyPEDS with these properties is depicted in Figure 2.1, where each node represents a Mica-Z mote [3] and each link represents a bidirectional communication channel over a shared medium, e.g., the RF channel specified by IEEE 802.15.4.

Each role requires to perform different tasks. Therefore, resource consumption for each role is different and depending on the assigned role, resource consumption on some nodes may be higher than that on the other nodes. This means that if the nodes are always assigned to the same roles, some nodes may exhaust their limited resources earlier than the others and, consequently, after a certain time-period some parts of the network may disappear. Therefore, it is necessary to balance the resource consumption among the nodes in order to extend the lifetime of a wireless sensor network. This can be achieved by changing the roles of the nodes over the time dynamically. For this purpose several protocols have been proposed such as *Low Energy Adaptive Clustering Hierarchy (LEACH)* [28] and *non-manipulable aggregator node election protocol* [58]. An aggregator node needs to perform not only an aggregator function f^1 , but also computationally expensive encryption functions. Thus, the resource consumption on aggregator nodes is particularly higher and the

¹ $f : \underbrace{\{0, 1\}^k \times \dots \times \{0, 1\}^k}_{x \text{ times}} \rightarrow \{0, 1\}^{k+l}$ where $k + l$ is much smaller than $x \cdot k$.

aggregator role in the network should be distributed equally. The appropriate election protocol should fulfill the following requirements.

- For any *epoch* t , one node is responsible only for one role, whereby t denotes the lifetime of a sensor network.
- For two subsequent epochs, the roles assigned to one node should be different. This means for two epochs t and $t + 1$, the roles should be distributed such that $r_n^t \neq r_n^{t+1}$ holds, whereby r_n^t represents the role of node n in epoch t .

In order to find the promising aggregator node election protocol, which meets the requirements stated above, Sirivianos *et al.* studied the requirements for a *non-manipulable aggregator node election protocol* in [58]. In contrast to existing node election protocols such as LEACH [28], HEED [65], and VCA [52], the non-manipulable aggregator node election protocol is not influenced and disrupted by the flow of the information produced within the network. Therefore, an election protocol like that may be employed to flat the overall resource consumption and, accordingly, extend the lifetime of a wireless sensor network.

The data transmission towards the reader device occurs in hop-by-hop manner, where every node which is not in sleeping mode may become a hop. Therefore, the data transmission may be still possible, even if some nodes are exhausted. The transmission may be performed in two ways. When data is requested, either each sensor node transmits its stored data directly to the reader device over forwarder nodes or they send the data to an intermediate aggregator node where the data from several sensor nodes is aggregated and then transmitted. The advantage of the later approach is that the amount of the data being transmitted is reduced. Both approaches are introduced in Section 2.7.1 and 2.7.2, respectively.

In conclusion, the network model of TinyPEDS is an asynchronous wireless sensor network, in which the roles assigned to the nodes change dynamically over the time to balance the load within the network. Therefore, in this network model the nodes require not only to run routing or data processing related applications, but also applications for minimizing the resource consumption in the network.

2.3 Threat models

Generally, an asynchronous wireless sensor network provides the following functionalities.

- Sensing and collecting data
- Processing and transmitting the sensed data
- Storing the sensed data
- Providing the sensed data to a reader device

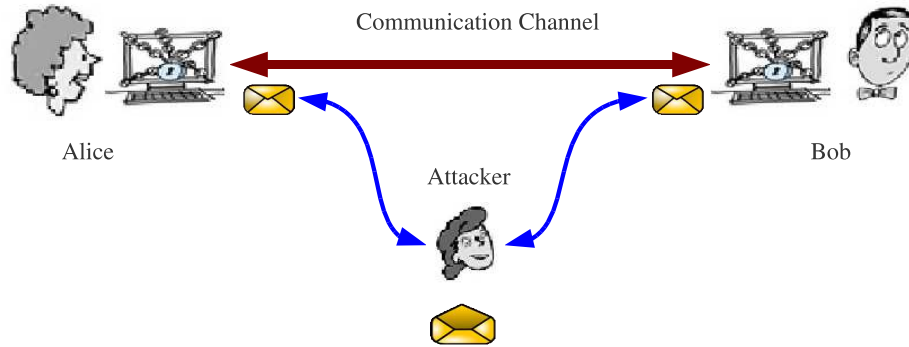


Figure 2.2: Dolev-Yao threat model

Since a wireless sensor network is generally utilized in a public environment, these functionalities imply that not only the data stored in the network, but also the communication between the nodes and the reader device have to be secured against attacks. Therefore, there is a need for a standard model for adversaries in wireless sensor networks.

In the areas of computer security and cryptography standard model for adversaries is the *Dolev-Yao threat model* [20]. However, this model does not reflect all of the security problems of wireless sensor networks. Thus, in the following, not only the Dolev-Yao threat model is presented, but also its wireless sensor network adapted version, namely the *extended Dolev-Yao threat model*.

2.3.1 Dolev-Yao threat model

The Dolev-Yao threat model is presented in [40]. A communication network such as the internet is an open network. This means that an attacker can join such a network for sending messages to or receiving messages from other entities in the network. Therefore, it is assumed that in such open networks, an attacker is able to duplicate, delete, eavesdrop, and inject messages. The Dolev-Yao threat model, depicted in Figure 2.2, assumes that two communicating entities, e.g. Alice and Bob, communicate over an insecure communication channel and the attacker has the following characteristics.

- The Attacker can represent herself as Alice or Bob.
- The Attacker can receive any message sent by Alice or Bob.
- The Attacker can communicate with Alice or Bob.
- The Attacker can send messages as representing herself as Alice or Bob.

Therefore, in the Dolev-Yao threat model any messages sent to the network is assumed to be received by the attacker and, consequently, any message received from the network is considered being modified by the attacker.

On the other hand, this model assumes that the attacker can not do the following things.

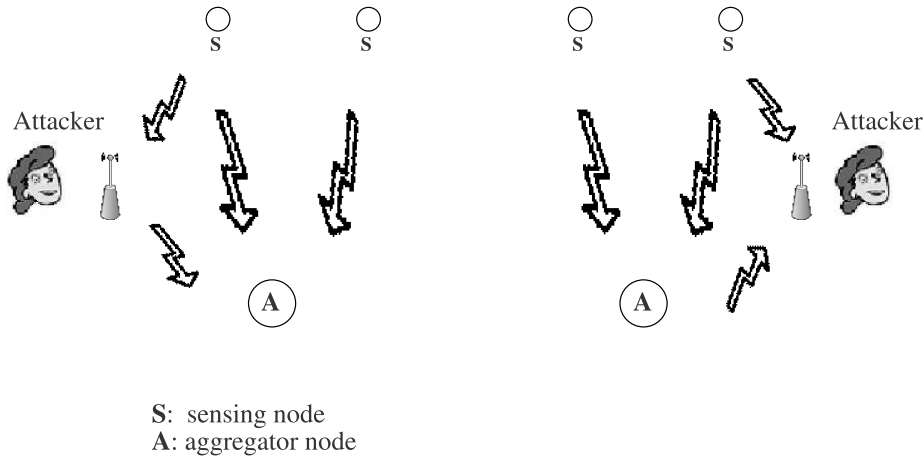


Figure 2.3: Extended Dolev-Yao threat model

- The attacker can not guess a random number chosen by Alice or Bob.
- The attacker can not decrypt the encrypted messages without corresponding private keys.
- The attacker can not access the computer of Alice or Bob, meaning that he can not find the private data such as private keys stored in Alice's or Bob's computer.

In conclusion, the Dolev-Yao threat model assumes that the channel, over which the entities communicate, is insecure. For this reason, not only the messages over that channel may be intercepted, manipulated, and deleted, but also new messages may be added to the communication. However, the Dolev-Yao model implicitly supposes that the communicating end-points are assumed to be secure against any type of attacks.

2.3.2 Extended Dolev-Yao threat model

In wireless sensor networks the extended Dolev-Yao threat model, depicted in Figure 2.3, may be used as the standard model for adversaries. In this model adversaries use the security problems stemming from the fact that wireless sensor networks are typically employed in a public environment and the hardware of the nodes is usually not tamper-resistant. Accordingly, the attacker has the following characteristics.

1. The attacker may eavesdrop the data transmission between sensor nodes.
2. The attacker may inject encrypted or non-encrypted data to the wireless communication between sensor nodes.
3. The attacker may pick one or more sensor nodes and retrieve sensitive information stored on them.
4. The attacker may break an encryption algorithm by using typical attacks such as *known plaintext attack*.

5. The attacker may decrypt an encrypted data, if she has a corresponding secret key.

Eavesdropping (1) on the wireless communication between nodes is trivial. An attacker requires only a laptop with a wireless communication capability and some wireless networking tools such as a packet sniffer. In the similar way, an attacker may inject any kind of messages to the wireless communication between the nodes (2). Since the nodes are typically spread over a public environment without any protection against the physical access, an attacker can pick the nodes from a wireless sensor network (3). If the transmitted data is encrypted with a *symmetric cryptosystem*, an attacker can perform the known plaintext attack (4). More precisely, assume that a sensor node is responsible for measuring the temperature from the environment. Firstly, an attacker eavesdrops the data transmission between the nodes, e.g. between a sensor node and an aggregator node, to get some samples of ciphertexts. In order to gain some plaintexts, he or she senses the temperature of the environment. After collecting some cipher and plaintext pairs the attacker can perform the known plaintext attack to retrieve the corresponding symmetric secret key. Once the attacker gets the secret key, he or she can decrypt every message encrypted with that key (5).

In order to limit all these possible damages, two approaches, namely using *tamper-resistant hardware* and *probabilistic security*, may be employed in TinyPEDS.

- **Using tamper-resistant hardware:**

If the nodes are equipped with tamper-resistant hardware, an attacker may be prevented from extracting data including sensible information such as secret keys. However, tamper-resistant hardware units are generally assumed to be expensive. Thus, by employing such a hardware the total cost of the wireless sensor network may become uneconomical compared to the value of the information retrieved by it. Furthermore, this approach does not solve the security problems (i.e. 1, 2, 4, and 5) stemming from the insecure wireless communication between the nodes within the network. Therefore, an additional approach providing a *probabilistic security* is necessary to make the sensor networks secure against the possible attacks.

- **Probabilistic security:**

As motivated from the extended Dolev-Yao threat model, the data provided by sensor networks has to be secured against attackers. The probabilistic approach requires an encrypted data storage and data transmission. However, as stated in the extended Dolev-Yao threat model, it is assumed that an attacker may still decrypt the data, even it is encrypted. Therefore, in the context of wireless sensor networks the probabilistic security assumes that an attacker attends to break an encryption only when the value of the data or damage, which is obtained in the case of a successful attack, is higher than the efforts required for performing the attack.

The resource consumption of the encryption schemes depends on several factors such as key length and the type of the scheme. Compared to symmetric key encryption schemes, public key schemes are in general more expensive with

respect to resource consumption. Therefore, employing only a public key encryption scheme for securing wireless sensor networks is not feasible. However, the major disadvantage of symmetric encryption schemes is that secret keys has to be stored on the nodes. Thus, since the nodes are usually equipped with non-tamper-resistant hardware, an attacker may retrieve the symmetric secret key by capturing the nodes. However, even if an attacker retrieves the secret key, the data captured by him/her is limited to the subset of data, which is encrypted with the gained secret key. Therefore, as motivated from the probabilistic approach, TinyPEDS employs a symmetric encryption scheme such that each nodes stores a different symmetric secret key, which is only shared with the reader device. Accordingly, even if the attacker breaks one node's secret key, the attacker's information gain is limited only to the data processed on that node.

For public key encryption schemes there is no need to store the secret key on the nodes. Thus, an attacker may not obtain any sensible information, even if he/she picks up a node and reads its internal states. For this reason TinyPEDS encrypts a large amount of data by using an asymmetric encryption scheme in order to defend the probabilistic security of a wireless sensor network.

In conclusion, although symmetric encryption schemes are computationally cheaper compared to public key encryption schemes, asymmetric encryption schemes offer a better system security, because no secret key storage on the sensor nodes is required. However, the size of the ciphertexts from asymmetric encryption schemes is generally larger than that of symmetric encryption schemes. Therefore, with respect to the resource consumption, e.g transmission cost, the use of an asymmetric scheme is a disadvantage. Moreover, due to the hardware restrictions of the employed nodes, using only an asymmetric encryption scheme is not feasible. As solution for these problems, TinyPEDS proposes a trade-off between the security level and resource consumption. Therefore, a subset of the data, that is representing very limited amount of information, is encrypted with a relatively weaker symmetric encryption, while the data representing the information from large regions or for a long-term storage is encrypted by using an asymmetric encryption scheme. The encryption schemes employed in TinyPEDS are studied in Section 2.5.

2.4 Concealed data aggregation

As resource consumption of the nodes is a critical factor for the overall lifetime of a wireless sensor network, it is necessary to employ techniques to reduce it. The in-network data aggregation works toward this goal by reducing the amount of the data being stored and transmitted, while still providing an appropriate degree of information to the reader device. Figure 2.4 depicts an example for the in-network data aggregation process where the sensed values are not encrypted. However, as motivated from the introduced security problems, the data within wireless sensor networks has to be encrypted in order to limit damage caused by attackers. In this case, the aggregator node A must calculate the $Sum = [Enc(15) + Enc(16) + Enc(18) + Enc(14)]$ of the encrypted values. This scenario is depicted in Figure 2.5. The problem with adding ciphertexts is that $Dec[Enc(15) + Enc(16) + Enc(18) + Enc(14)]$ is not equal to $(15 + 16 + 18 + 14)$. This means that the classical additive operation $+$ is not

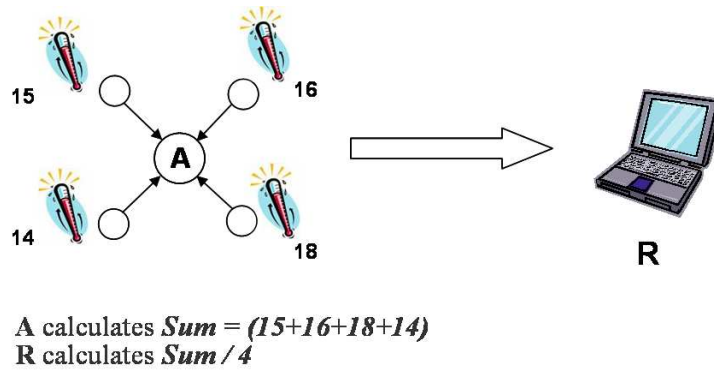


Figure 2.4: Data aggregation in wireless sensor networks

applicable over ciphertexts of conventional encryption schemes. Thus, before adding them, the encrypted values have to be transformed such that additive operation on encrypted values is defined. Such an encryption scheme is called *additive homomorphic encryption scheme*. Two additive homomorphic encryption schemes are applied in TinyPEDS and they are introduced in the following section.

2.5 Encryption Schemes

TinyPEDS proposes a *concealed in-network data aggregation* not only for reducing the energy consumption tied to data transmission, but also for increasing system and data security in a multi-hop asynchronous wireless sensor network. Therefore, an appropriate symmetric and public key encryption scheme to be employed in TinyPEDS should fulfill certain requirements which are summarized subsequently. Note that the reference for this section is [46] and [23].

2.5.1 Requirements for an appropriate encryption scheme

The requirements which an appropriate encryption solution has to fulfill may be divided into tree classes, namely requirements for a concealed data aggregation, system and data security, and lifetime of the wireless sensor network.

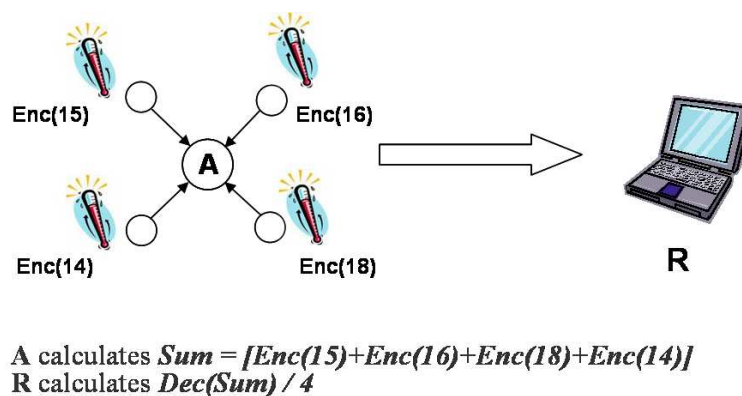


Figure 2.5: Concealed data aggregation in wireless sensor networks

- **Requirements for a concealed data aggregation:**

The concealed data aggregation requires to add ciphertexts. Therefore, an appropriate encryption scheme should be *additive homomorphic*. This means that it needs to support the property such that the following equation holds.

$$Enc(a_1 + a_2 + \dots + a_n) = Enc(a_1) \diamond Enc(a_2) \diamond \dots \diamond Enc(a_n)$$

Thereby, $Enc(a)$ denotes the encryption of a message a and \diamond represents an additive operation performed over ciphertexts from a symmetric or public encryption scheme.

Since TinyPEDS proposes to employ both a symmetric and a public key encryption scheme (See Section 2.3.2), two additive homomorphic encryption schemes are needed, namely *symmetric additive homomorphic encryption* and *asymmetric additive homomorphic encryption*. Note that within this thesis PH_s and PH_a denote a symmetric additive homomorphic encryption and an asymmetric additive homomorphic encryption, respectively, and are defined as follows:

- **Definition 2.1 Symmetric additive homomorphic encryption**

Let be $Enc : K \times P \rightarrow C$ and $Dec : K \times C \rightarrow P$, where Enc is a symmetric encryption function, Dec is the corresponding decryption function, P is the message space, C is the ciphertext space, and K is a set of symmetric secret keys.

Enc is additively homomorphic, if and only if for given messages $a_1, a_2, \dots, a_n \in P$ and for secret keys $k_1, k_2, \dots, k_n \in K$, there exists a key $k \in K$ such that the following equation holds:

$$(a_1 + a_2 + \dots + a_n) = Dec_k(Enc_{k_1}(a_1) \oplus Enc_{k_2}(a_2) \oplus \dots \oplus Enc_{k_n}(a_n)) \quad (2.1)$$

Note that the symbol $+$ denotes an addition operation on the elements of the message space and the symbol \oplus represents a corresponding additive operation on the elements of the ciphertext space.

- **Definition 2.2 Asymmetric additive homomorphic encryption**

Let be $Enc : K_{pk} \times P \rightarrow C$ and $Dec : K_{sk} \times C \rightarrow P$, where Enc is an asymmetric encryption function, Dec is the corresponding decryption function, pk is a public key, sk is a corresponding private(secret) key, which is stored only on the reader device, P is the message space, and C is the ciphertext space.

Enc is additively homomorphic, if and only if for given messages $a_1, a_2, \dots, a_n \in P$ and for a key pair $(pk, sk) \in (K_{pk}, K_{sk})$ the following equation holds:

$$(a_1 + a_2 + \dots + a_n) = Dec_{sk}(Enc_{pk}(a_1) \boxplus Enc_{pk}(a_2) \boxplus \dots \boxplus Enc_{pk}(a_n)) \quad (2.2)$$

Note that the symbol $+$ stands for an addition operation on the elements of the message space and the symbol \boxplus represents a corresponding additive operation on the elements of the ciphertext space.

- **Requirements for system security and data security:**

An appropriate encryption scheme should have or enable the following properties. Firstly, in order to prevent attackers from unauthorized information gain, it should be provable secure. In other words, the security level provided by the selected encryption scheme should be based on the difficulty of solving a computational hard problem. Secondly, it should not require to store sensitive key material on the nodes. Finally, the key management of the employed encryption scheme should be simple, meaning that no bandwidth intensive techniques are needed to identify the encryption keys being used by the nodes.

- **Requirements for moderate wireless sensor network lifetime:**

The overall lifetime of a wireless sensor network should be moderately long. Accordingly, an optimal encryption scheme should ensure that cryptographic operations required for the encryption should not be computationally expensive. Furthermore, the size of ciphertexts produced by the employed encryption scheme should not be very large. Otherwise, the transmission of encrypted data results in an increased energy consumption and thus decreases the overall lifetime of the wireless sensor network.

2.5.2 Selecting an appropriate encryption scheme

There are several encryption schemes in literature which meet some or all of the desired criteria outlined in the previous subsection. However, resource consumption of those candidates differ. For selecting the most promising encryption scheme they should be analyzed with respect to their computational efficiency, energy, and storage consumptions. In the following subsections the candidate homomorphic encryption schemes are studied.

2.5.2.1 Symmetric encryption scheme candidates

In this subsection two symmetric additively homomorphic encryption schemes proposed in [21] and [13] are introduced.

- **Additive homomorphic encryption scheme proposed in [21]:**

In [21] Domingo-Ferrer proposed a provably secure additive privacy homomorphism which fulfills the equation 2.1.

In this PH_s scheme prior to performing the encryption, a message $a \in [-m', m']$ is randomly divided into secrets $(a_1, a_2, \dots, a_d) \in [-m, m]$ such that $a = \sum_{j=1}^d a_j \pmod{m'}$. The encryption function Enc_k and the corresponding decryption function Dec_k are defined as follows:

- **Encryption:**

$$c = Enc_k(a) = (a_1 r \pmod{m}, a_2 r^2 \pmod{m}, \dots, a_d r^d \pmod{m}) \quad (2.3)$$

- **Decryption:**

$$(a_1, a_2, \dots, a_d) = (r^{-1} \pmod{m}, r^{-2} \pmod{m}, \dots, r^{-d} \pmod{m}) \quad (2.4)$$

$$a = Dec_k(c) = \sum_{j=1}^d a_j \pmod{m'} \quad (2.5)$$

Thereby, $d \geq 2$ and m is such an integer that it has not only many small divisors, but also there are many integers which are smaller than m and can be inverted modulo m . The parameters d and m are *public*. The integer pair $k = (r, m')$ denotes the *secret key pair*. Note that this secret integer pair is selected such that $r^{-1} \bmod m$ exists and the result of $\log_{m'} m$ is an integer, whereby m' is a small divisor of m .

One major disadvantage of this scheme is that the secret key shared among all sensor nodes and the reader device is the same. This means that if an attacker retrieves the secret key by e.g. compromising the nodes, he/she can access a large amount of data. Obviously, this breaches the probabilistic security approach proposed in TinyPEDS. Another disadvantage of this PH_s scheme is its weak security level. Although Domingo-Ferrer showed that the proposed homomorphic encryption scheme is provably secure against *known-plaintext attacks*, it provides weak level of security against *ciphertext-only attacks* [46].

- **Additive homomorphic encryption scheme proposed in [13]:**

In this approach instead of using a single shared secret key, each node has its own unique secret key, which is only known to the reader device. The advantage of this multi-key approach is that even if an attacker obtains a secret key by e.g. compromising a node, the information gain is limited to the amount of data, which is stored on that single corresponding node. Thus, this PH_s scheme fulfills the probabilistic security required for TinyPEDS.

Castelluccia *et al.* define the encryption and decryption functions Enc_k and Dec_k as follows.

– **Encryption:**

$$c = Enc_k(a) = a + k \bmod m \quad (2.6)$$

– **Decryption:**

$$a = Dec_k(c) = Enc_k(a) - k \bmod m \quad (2.7)$$

Thereby, $0 \leq a < m$, $0 \leq k < m$, and the modulus m is a large integer. Note that k represents the symmetric secret key, while a denotes the message to be encrypted.

This scheme provides a desired additive homomorphism over encrypted values, since the following equation always holds.

$$\begin{aligned} Dec_k(c_1 + c_2 + \dots + c_n) &= Dec_k(Enc_{k_1}(a_1) + Enc_{k_2}(a_2) + \dots + Enc_{k_n}(a_n)) \\ &= (a_1 + a_2 + \dots + a_n) \bmod m \end{aligned}$$

with $(k = k_1 + k_2 + \dots + k_n) \in [0, m - 1]$ and $a_1, a_2, \dots, a_n \in [0, m - 1]$.

However, for this encryption scheme the modulus m should be carefully selected. More precisely, if the sum of n ciphertexts, i.e. $sum = (c_n + c_{n-1} + \dots + c_2 + c_1) = (Enc(a_n) + Enc(a_{n-1}) + \dots + Enc(a_2) + Enc(a_1))$ is larger than m , then the decryption of sum yields a value that is smaller than the actual value that is $(a_n + a_{n-1} + \dots + a_2 + a_1)$. Therefore, m should be selected such

that $m = 2^{\lceil \log_2(p*n) \rceil}$, when the homomorphic encryption is required to be performed over n messages and the maximum of those messages are represented by p . This PH_s scheme is proven to be perfectly secure in [13].

Since this PH_s scheme requires the secret key to be stored on the nodes, it does not fulfill all security requirements. However, as already stated in Section 2.3.2, due to its high computation and bandwidth efficiency, it is employed for encrypting a very small subset of data.

In conclusion, the Domingo-Ferrer's scheme not only provides weaker system security, but also its implementation is more complex, which may result in increased resource consumption. Therefore, TinyPEDS employs the symmetric encryption scheme proposed by Castelluccia *et al.*.

2.5.2.2 Asymmetric encryption scheme candidates

There are several candidates for the PH_a which fulfills the requirement of Equation 2.2. In [46] Mykletun *et al.* studied a selected subset of the public key encryption schemes, which meet a large portion of the requirements introduced in Section 2.5.1. A subset of those public key cryptosystems are the *elliptic curve Naccache-Stern (EC-NS) scheme*, the *elliptic curve Okamoto-Uchiyama (EC-OU) scheme*, the *elliptic curve Pailler (EC-P) scheme*, and the *elliptic curve ElGamal (EC-ElGamal) scheme*. The first three schemes are described by Pailler in [51] and exhibit the elliptic curve variants of those algorithms proposed originally by Naccache-Stern [47], Okamoto-Uchiyama [48], and Pailler [50]. The EC-ElGamal encryption scheme is proposed by Mykletun *et al.* in [46].

Table 2.1: Comparison of public key encryption scheme candidates [46]

Scheme	Encryption	Addition	Bandwidth
EC-NS	$\frac{15}{2} n $	1800	$5 n $ 5 $ n + 2$ 1026
EC-OU	$5 + \frac{15}{2} m + \frac{15}{2} 2k $	690	$5 n $ 5 $ n + 2$ 1026
EC-P	$\frac{15}{2} n^2 $	33105	5 20 $2 n + 4$ 2052
EC-ElGamal	$2\frac{15}{2} p + \frac{15}{2} m $	38	$10 p $ 1 $2(p + 1)$ 328

Table 2.1 represents the performance of the public key homomorphic encryption candidates in terms of their computational costs for encryption, addition of two ciphertexts, and their ciphertext sizes. Remember that sensor nodes do not have to decrypt the encrypted values, as the decryption of the aggregated data is performed on the reader device, which is assumed to have unlimited resources. Therefore, the computational costs for decryption are ignored and not shown in Table 2.1. All table entries consist of two columns. The values in the left side columns denote the formulas used to determine the respective costs which are represented in the right side columns. The formulas refer to parameters of the respective schemes. The size of the p for EC-ElGamal is 163-bit, while that for EC-OU is 341-bit. In order to compare all computations equitably the respective costs for encryption and addition are converted and measured in terms of 1024-bit modular multiplications. Thus, the n in Table 2.1 refers to 1024-bit integers.

A careful analysis of Table 2.1 shows that the EC-ElGamal encryption scheme is the most promising scheme to be used in TinyPEDS, due to its superior performance and smaller ciphertext sizes.

- **Elliptic curve ElGamal encryption scheme (EC-ElGamal):**

Since the original ElGamal encryption scheme, proposed in [22], is *multiplicative homomorphic*, it can not be directly employed in TinyPEDS which requires an additive homomorphic encryption. Therefore, the ElGamal encryption scheme must be transformed to an additive group. Algorithm 1 represents the EC-ElGamal encryption scheme transformed to an additive group.

The function $map()$ is a deterministic mapping function used to map plaintext values into plaintext curve points M such that

$$map(m_1 + m_2 + \dots + m_n) = \underbrace{map(m_1)}_{M_1} + \underbrace{map(m_2)}_{M_2} + \dots + \underbrace{map(m_n)}_{M_n}$$

holds, whereby $m_1, m_2 \in GF(p)$. Since the addition operation over an elliptic curve requires both operands to be on that curve, prior to performing an addition of two integers, they should be mapped to the corresponding elliptic curve points. This explains why the mapping function is necessary. As proposed in [6] the homomorphic mapping function used in TinyPEDS is based on using multiples of the generator point G of the elliptic curve. This means that the mapping function converts a plaintext m to the point mG . The reverse mapping function $rmap()$ then extracts m from a given point mG . The mapping function, namely

$$map : m \rightarrow mG \text{ with } m \in GF(p) \quad (2.8)$$

fulfills the required homomorphic property due to the fact that the equation

$$\begin{aligned} M_1 + M_2 + \dots + M_n &= map(m_1 + m_2 + \dots + m_n) \\ &= (m_1 + m_2 + \dots + m_n)G \\ &= m_1G + m_2G + \dots + m_nG \end{aligned}$$

holds with $m_1, m_2, \dots, m_n \in GF(p)$, the generator point G , and the modulus p .

The mapping function is not security relevant, since it only converts an integer to an elliptic curve point. This means, it neither increases nor decreases the security of the EC-ElGamal encryption scheme. Note that the reverse mapping function is the same as solving the *discrete logarithm problem* over an elliptic curve and, therefore, a weakness of this scheme. However, since the mapping function is only performed on the reader device, which is assumed to have unlimited resources, this disadvantage does not affect the performance and resource consumption within the network.

In conclusion, according to the analysis made in [46], the EC-ElGamal scheme becomes the most promising candidate for using in TinyPEDS, because of its efficiency both in computation and bandwidth. However, the main disadvantage of this scheme

Algorithm 1 Elliptic curve ElGamal encryption scheme from [46]

Private key: $x \in GF(p)$
Public key: E, p, G, Y , whereby $Y = xG$ and elliptic curve E over $GF(p)$ with $G, Y \in E$
Encryption (enc()): For a given plaintext $m \in [0, p - 1]$ and random $k \in [1, n - 1]$, where n is the order of E .

$$M = \text{map}(m)$$

$$\text{ciphertext } C = \text{enc}(m) = (R, S) = (kG, M + kY)$$

Decryption (dec()):

$$M = \text{dec}(C) = \text{dec}(R, S) = -xR + S$$

$$m = \text{rmap}(M)$$

is that the reverse mapping function required during decryption may be in some cases too costly. However, since the number of the aggregated values is limited and the maximum length of the final aggregation is assumed to be at most three bytes, see [46], the reverse mapping of the point mG with 24-bit m can be calculated on the reader device fast enough.

2.6 Distributed storage of encrypted and aggregated data

TinyPEDS aims at providing encrypted and aggregated collaborative distributed data storage in asynchronous wireless sensor networks. Such a data storage has the following important advantages. Firstly, Aggregator nodes do not need to decrypt the encrypted data received from neighbor nodes for performing the data aggregation over them. Thus, the number of computations required on the nodes is reduced. Secondly, sensor nodes do not need to store secret keys from other sensor nodes, but only its own secret key. Therefore, the overall system security of a wireless sensor network is increased. Thirdly, TinyPEDS employs in-network data processing that reduces the amount of the data to be transmitted, therefore, the transmission cost for distributed data storage is minimized. Fourthly, by employing election protocols such as the LEACH and the non-manipulable aggregator node election protocol, the roles and the data storage within the network is distributed over multiple sensor nodes. Thus, the resource consumption of the network is balanced. Furthermore, since the data is replicated on multiple nodes, even if parts of the network are exhausted, the lost information may be restored in most cases (See Section 2.8.1). Finally, TinyPEDS approach applies the PH_s and PH_a to aggregate encrypted data to ensure higher system security. Note that the notation used in this section is similar to that used in [23].

Prior to explaining how TinyPEDS performs encrypted and aggregated collaborative distributed data storage, the notation is introduced in the following. In TinyPEDS each node may act as a forwarder, aggregator, sensor, or idle node. Since the storage capacity of the nodes in the network is limited and aggregator nodes store the data received from its neighbors, the number of the neighbors, for which an aggregator node is responsible, should be limited. Thus, the wireless sensor network is divided into quarters. Let Q_z represents the z 'th quarter of the wireless sensor network and the role of the nodes in this quarter are represented by $Q_z r^t$ with $r \in \{a, f, s, i\}$.

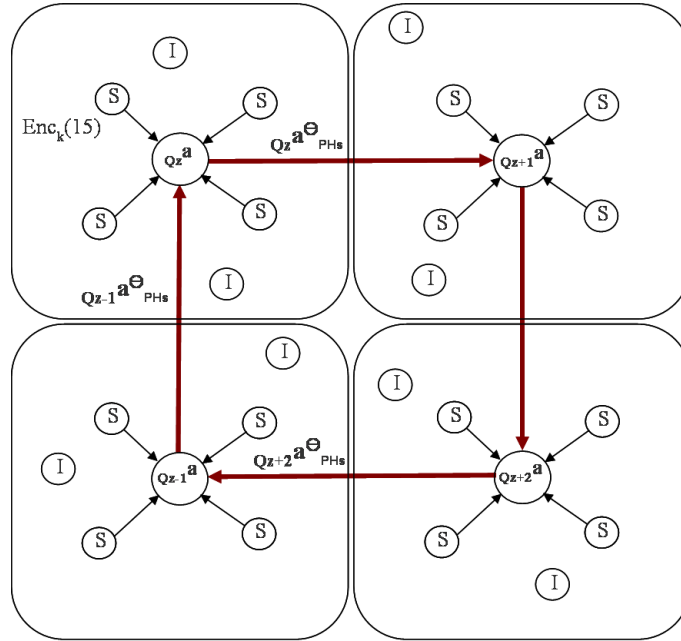


Figure 2.6: TinyPEDS cluster structure [23]

Thereby, a, f, s, i denotes aggregator, forwarder, sensing, and idle node, respectively. Figure 2.6 depicts a graphical representation of such a wireless sensor network with four quarters. Even though the number of sensor nodes owned by each quarter in Figure 2.6 is the same, this is not the case in real applications. For larger wireless sensor networks, it is even necessary to divide quarters into subquarters. Such a network is called a *hierarchical wireless sensor network* [23].

In TinyPEDS encrypted and aggregated collaborative distributed data storage is obtained as follows. Firstly, for every epoch t the node election protocol, e.g. non-manipulable aggregator node election protocol, elects an aggregator node $Q_z a^t$ for every quarter. In addition to selecting aggregator nodes other roles are distributed among all other nodes such that the load of the network is balanced. Thus, each node takes a role such that in every epoch t a node $Q_z a^t$ is responsible for aggregating data received from its neighbors. A node $Q_z s^t$ is responsible for monitoring its environment and for transmitting monitored data to the aggregator of the quarter Q_z after encrypting it with PH_s . A node $Q_z f^t$ acts as a hop in hop-by-hop data transmission. Finally, A node $Q_z i^t$ is not available in the epoch t . Since in any epoch t a sensor network continuously monitors a region, the epoch t consists of a number of time slots τ , after which every node performs its assigned task. Therefore, the tasks being performed within the network are divided into two groups. Firstly, the tasks performed in each time slot between τ and $\tau + 1$ which is depicted by θ and, secondly, the tasks performed at the end of each epoch t . They are performed as follows.

1. Performing the tasks in a time slot θ with $\tau \leq \theta \leq \tau + 1$:
 - (a) Each sensor node $Q_z s^t$ monitors the region, for which it is responsible and encrypts that monitored data by using the symmetric homomorphic encryption scheme PH_s . Thereafter, $Q_z s^t$ transmits the encrypted data to

its aggregator, if the transmission distance between it and the aggregator is one hop. Otherwise, it transmits the encrypted data to the next $Q_z f^t$.

- (b) Each forwarder node $Q_z f^t$ transmits the data received from the previous hop to its aggregator $Q_z a^t$. If its neighbor is also a forwarding node, it sends the data to the next forwarding hop. The data is transmitted towards the aggregator node of the epoch t in a hop-by-hop manner recursively until the aggregator node is reached.
- (c) Each aggregator node $Q_z a^t$ adds the encrypted data received from sensor nodes by using the symmetric additively homomorphic operation \oplus and stores it persistently. Note that since the reader device may demand an aggregated value from a certain time slot θ , it stores the data such that it later can retrieve the data for a given time slot. From now on, this value is represented by $Q_z a_{PH_s}^\theta$.
- (d) Since an idle node $Q_z i^t$ is in sleeping mode, it is not available during epoch t .

Above operations are performed repeatedly until the epoch t is finished.

2. Performing the tasks at the end of each epoch t :

- (a) At the end of each epoch t , as depicted Figure 2.6 the aggregator node $Q_z a^t$ of each quarter transmits the aggregation $Q_z a_{PH_s}^\theta$ of the monitored data to its aggregator node, namely the aggregator node $Q_{z+1} a^t$ of the next quarter Q_{z+1} .
- (b) Each aggregator node $Q_z a^t$ adds the data $Q_{z-1} a_{PH_s}^\theta$ that is received from the aggregator node $Q_{z-1} a^t$ with its own aggregation, namely $Q_z a_{PH_s}^\theta$, by using the symmetric additive homomorphic operation \oplus . Thereafter, it encrypts that data by using the selected asymmetric homomorphic encryption scheme PH_a . From now on, this value stored persistently is denoted by $Q_z a_{PH_a}^\theta$ with $Q_z a_{PH_a}^\theta = Enc_{pk}(Q_z a_{PH_s}^\theta \oplus Q_{z-1} a_{PH_s}^\theta)$.
- (c) Moreover, each aggregator node $Q_z a^t$ stores the data $Enc_{pk}(Q_z a_{PH_s}^\theta) \parallel Q_z a_{PH_a}^\theta \parallel (Q_z a_{PH_a}^\theta \boxplus Q_z a_{PH_a}^{\theta-1}) \parallel (Enc_{pk}(Q_z a_{PH_s}^\theta) \boxplus Enc_{pk}(Q_z a_{PH_s}^{\theta-1}))$ persistently. Note that \parallel denotes the concatenation of two encrypted messages. Since the reader device can demand an aggregated value from a certain time slot θ , each aggregator node stores the data such that they can later retrieve the data for any time slot θ . This storage is denoted by $Q_z a_{storage}^\theta$.

Above operations defined are performed repeatedly for each epoch t .

2.7 Query flooding and query response

This section explains how the reader device requests data from the wireless sensor network and how the TinyPEDS architecture handles these data requests.

2.7.1 Controlled query flooding

In order to obtain the monitored data from any part of a wireless sensor network, the reader device is placed in the center of a network and a database query is sent

to the network. Locating the reader device in the center is not a must but the most energy-efficient way. The database query should contain certain information such as the region from which the data is requested, a time interval in which a region is monitored, a query-range indicating the flooding range of the query, and the query type. The type of a query QT may be C or D . C indicates a *continuous database query*, while a *disaster database query* is represented by D . Note that the disaster query is employed in *disaster model*, which is explained in Section 2.8.1. The main difference between these two query types is that the disaster query floods the whole wireless sensor network where the data is replicated, while the continuous database query floods only the target region from where the information retrieval is desired.

The query sent from a reader device is represented as follows.

$$query = (region, duration, aggregation, TTL, QT) \text{ ([23])}$$

Thereby, the parameter *region* denotes the quarter of the wireless sensor network from which the data retrieval is desired and *duration* represents the time interval in which the data was monitored. The query-range is denoted by the *TTL (time-to-live)* parameter, which represents the maximum hop-distance the query will travel within the wireless sensor network. The type of a query is denoted by the parameter QT . This means that if $aggregation \in \{<, >\}$, the data representing the response for the query towards the reader device is aggregated by comparison. Similarly, if the type of the aggregation is $+$, the data towards a reader device is aggregated by addition. As already introduced in Section 2.4, the aggregator function based on $+$ needs to be additive homomorphic. Analogously, there is a need for a privacy homomorphism which is homomorphic with respect to the comparison. Since the *comparative homomorphic privacy homomorphism* is not within the scope of this thesis, the reader is referred to [23] for detailed information about this privacy homomorphism.

Algorithm 2 Continuous database query [23]

```

1: if  $s \in Q_z$  and  $Q_z \subseteq region$  then
2:   if  $tll_{current} > 1$  then
3:      $tll_{current} = tll_{current} - 1$ 
4:      $s \rightarrow * : (region, [\tau_x, \tau_y], aggregation, tll_{current}, C)$ 
5:     if  $aggregation = true$  and  $Q_z a_{PH_a}^\theta \in Q_z s_{storage}^\theta$  and  $\tau_x \leq \theta \leq \tau_y$  then
6:        $s \rightarrow R : (Q_z s_{storage}^\theta)$ 
7:     end if
8:   end if
9: else
10:   $tll_{current} = 0$ 
11: end if

```

Algorithm 2 represents how a sensor node s processes a continuous database query

$$query = (region, [\tau_x, \tau_y], aggregation, tll_{current}, C)$$

requesting the environmental data monitored in a time interval $[\tau_x, \tau_y]$ from a quarter Q_z of a wireless sensor network.

In step 1 the sensor node checks, if the query is addressed to it or not. In the latter case, it ignores the query, otherwise, it checks if the maximum query range is achieved or not, which is shown in step 2. If it is within the query-range, it reduces the value of *TTL* (*time-to-live*) parameter by one, and broadcasts the modified query to all its neighbors. This transmission is performed in step 4 of the algorithm. Subsequently, in step 5 the node checks its persistent storage space for data, which was monitored for the region in the time interval $[\tau_x, \tau_y]$. If such data is found, the sensor node transmits it towards the reader device in step 6. Note that the data is transmitted directly to the reader device without being processed on an intermediate aggregator node. The reason for this is that if the data would be sent to an intermediate aggregator node where they are aggregated, the aggregated data would require a large number of bytes for being represented. Since, the reader device needs to perform brute force attack over the aggregated data (See Section 2.5.2.2), the aggregated data consisting of a large number of bytes is not desired. However, this problem would be overcome, if the number of intermediate aggregator nodes is selected such that the aggregated data may be represented by a small number of bytes. The only requirement for this approach is then to estimate a moderate number of aggregators and to divide a wireless sensor network into quarters such that each of those intermediate aggregator node is responsible for a reasonable number of sensor nodes. Such an approach is described in Section 2.7.2.

Algorithm 3 represents how the sensor node s processes a disaster database query

$$query = (region, duration, aggregation, ttl_{current}, D)$$

requesting the environmental data monitored in a time interval $[\tau_x, \tau_y]$ from a quarter Q_z of a wireless sensor network. A disaster database query is sent only when the data retrieval with a continuous database query failed.

Note that in the case of hierarchical wireless sensor networks (Section 2.6), an aggregator of a higher level takes the role of the reader device and every sensor node transmits the requested data to that aggregator node. Therefore, depending on the query type, a query request is processed in each level in exactly the same way except for step 6 of Algorithm 2 and 3.

Algorithm 3 Disaster database query [23]

```

1: if  $s \in N^t \setminus Q_z$  then
2:   if  $ttl_{current} > 1$  then
3:      $ttl_{current} = ttl_{current} - 1$ 
4:      $s \rightarrow * : (region, [\tau_x, \tau_y], aggregation, ttl_{current}, C)$ 
5:     if  $aggregation = true$  and  $Q_z s_{PH\alpha}^\theta \in Q_z s_{storage}^\theta$  and  $\tau_x \leq \theta \leq \tau_y$  then
6:        $s \rightarrow R : (Q_z s_{storage}^\theta)$ 
7:     end if
8:   end if
9: else
10:   $ttl_{current} = 0$ 
11: end if

```

2.7.2 Aggregated data response

TinyPEDS employs the asymmetric homomorphic encryption scheme PH_a for encrypting the data at the end of each epoch. Thus, there is a possibility to perform in-network processing also during the data response phase.

In the *controlled query flooding* approach every sensor node transmits the requested environmental data directly to the reader device. The main disadvantage of this approach is that a huge amount of data needs to be transmitted in the network, leading to huge energy consumption. An alternative approach is to aggregate the data on the fly. In other words, some nodes in the network may aggregate the data before transmitting it towards the reader device. However, as stated in the previous section the number of the aggregator nodes should be selected such that the reverse mapping performed on the reader device can still be calculated with an acceptable effort. This selection can be made as follows. Assume that the environmental data monitored by each node can be represented in $\lceil \varpi/8 \rceil$ -byte and the number of the bytes for which a reverse mapping function can be performed with a moderate effort is ϑ -byte. Let further assume that a quarter of a wireless sensor network being monitored consists of n nodes. Under these assumptions, the number of the aggregator node x is selected such that $x = \lceil n/y \rceil$ is minimum, whereby $y(2^\varpi - 1) \leq 2^{\varpi \cdot \vartheta} - 1 < m$ and m is the modulus of the PH_s . More precisely, each aggregator node should be responsible for adding the data from y sensor nodes and the result of the additions should fit in ϑ bytes.

Another challenge is to decide the location of aggregator nodes. The optimal position of an aggregator node is located between the responding sensor nodes and the reader device. However, since finding an aggregator node with such a position is NP -hard, a heuristic method has to be employed [23].

Once an aggregator node a_{l-1} is elected, the data aggregation towards the reader device is performed as follows.

- Beginning from the lowest level, each sensor node s addressed by the query request from the reader device transmits the requested data, namely $Q_{l-1} s_{storage}^\theta$ with $\tau_x \leq \theta \leq \tau_y$, to the elected aggregator node of the lowest level a_{l-1} .
- The aggregator node a_{l-1} performs the homomorphic addition \boxplus over received environmental data from its neighbor sensor nodes. Let φ represents that aggregated data.
- If the aggregator node a_{l-1} has another aggregator node a_{l-2} from a higher hierarchy, it transmits φ to the aggregator a_{l-2} . Similar to a_{l-1} , a_{l-2} performs the homomorphic addition over received aggregated data from aggregator nodes located in a lower hierarchy by using \boxplus . Let ς represents that aggregated data. This step is performed recursively until the aggregator node in the highest level, namely a_{l-l} , receives and aggregates the requested data.
- The aggregator node a_{l-l} located in the highest hierarchy level transmits the aggregated data ς to the reader device.

2.8 Approaches for increasing data availability

2.8.1 Disaster model

Since clusters of sensor nodes may die because of disastrous events, this aspect has to be considered in order to increase the availability of the data stored in a wireless sensor network. For this purpose, TinyPEDS proposes to replicate the data monitored from a region in such a manner that the wireless sensor network can survive regional disasters.

In this model, it is assumed that the administrator of a wireless sensor network is immediately informed about the disaster. Once the administrator receives a disaster notification, he/she tries to get as much data from the network as possible. Furthermore, it is assumed that the maximum damage of possible disasters on the network can be estimated. This is important, since by using this estimation, the location of sensor nodes, which are not affected by the disaster can be estimated.

However, the main drawback of this model is an increased resource consumption. In other words, the data replication not only increases the storage and energy consumption of sensor nodes where the data is replicated, but also requires additional data transmission in the network. Thus, it is important that the data is only replicated as much as necessary making the data recovery after a potential disaster possible. The disaster model of TinyPEDS proposes that the data monitored from regions, which may be affected by possible disasters, should be replicated on the nodes, which are out of the disaster radius. More precisely, if the radius of a possible disaster is r , the distance between a pair of nodes needed for storing the replicated data should be at least $\gamma = 2r$, see [23]. As already mentioned above, the number of the replications

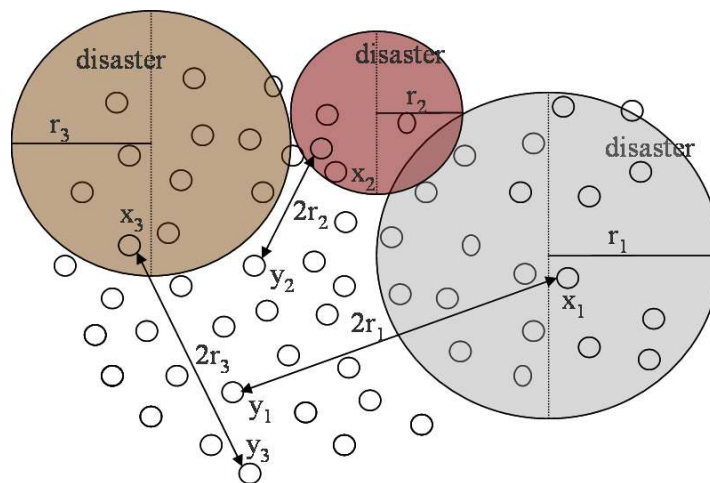


Figure 2.7: Disaster model

and the transmission distance effect the energy consumption of a sensor network. For minimizing the resource consumption, the data of a sensor node x , which may be affected by a possible disaster, is replicated only on one node y , which is not located in the radius of the disaster. The distance between the original and replication node should be at least $\gamma + \xi$, where ξ is as small as possible. This data replication ensures that the data replicated on y survives a possible disaster [23]. This behavior of the disaster model is depicted in Figure 2.7.

Remember that the query model for data restoration differs from the usual query model. Thus, the queries in a normal case, meaning that there is no disaster, are aimed at selecting the aggregated values from a specific time period and any region to receive their values, while a disaster query requests for any information that is remained in the network in order to salvage as much information as possible. Those query models are already introduced in Section 2.7.1.

2.8.2 Techniques for restoring the lost data

As stated in Section 2.6, the environmental data monitored from the quarter, e.g. Q_{z-1} , is not only stored on the aggregator node $Q_{z-1}a^t$, but also on the aggregator node $Q_z a^t$ of the quarter Q_z . This means that if the aggregator node $Q_z a^t$ for epoch t is exhausted or unavailable due to a disaster, the following techniques may be employed to recover the lost data:

• Data restoration technique 1:

Assume that the aggregator node $Q_z a^t$ from the quarter Q_z for the time slot $[\tau_x, \tau_y]$ with $\tau_x \leq \theta \leq \tau_y$ is exhausted. The environmental data $Q_z a_{PH_a}^\theta$ which is persistently stored on the aggregator node $Q_z a^t$ may be recovered as follows. Firstly, the reader device sends a disaster database query ($Q_z \cup Q_{z-1}, [\tau_x, \tau_y], +, ttl_{max}, D$) to obtain the environmental data from the remaining aggregator nodes, $Q_{z-1} a^t$, $Q_{z+1} a^t$, and $Q_{z+2} a^t$, for a time period $\tau \in [t_x, t_y]$. Secondly, the lost data may be recovered on the reader device by employing the following rules.

1. Aggregator nodes $Q_{z-1} a^t$, $Q_{z+1} a^t$, and $Q_{z+2} a^t$ transmit their persistent storage $Q_{z-1} a_{PH_a}^\theta$, $Q_{z+1} a_{PH_a}^\theta$, and $Q_{z+2} a_{PH_a}^\theta$ to the reader device.
2. Since $Q_z a_{PH_a}^\theta = Enc_{pk}(Q_z a_{PH_s}^\theta \oplus Q_{z+1} a_{PH_s}^\theta)$, the reader device decrypts those encrypted data received from aggregator nodes by using the decryption function D_{sk} for the PH_a to get the ciphertexts encrypted with the PH_s as follows.
 - (a) $ciphertext_1 = (Q_{z-1} a_{PH_s}^\theta \oplus Q_z a_{PH_s}^\theta) = Dec_{sk}(Q_{z-1} a_{PH_a}^\theta)$
 - (b) $ciphertext_2 = (Q_{z+1} a_{PH_s}^\theta \oplus Q_{z+2} a_{PH_s}^\theta) = Dec_{sk}(Q_{z+1} a_{PH_a}^\theta)$
 - (c) $ciphertext_3 = (Q_{z+2} a_{PH_s}^\theta \oplus Q_{z-1} a_{PH_s}^\theta) = Dec_{sk}(Q_{z+2} a_{PH_a}^\theta)$
3. Subsequently, the reader device recovers the lost information presented in $(Q_z a_{PH_s}^\theta \oplus Q_{z+1} a_{PH_s}^\theta)$ as follows.

$$Dec_k(Q_z a_{PH_s}^\theta \oplus Q_{z+1} a_{PH_s}^\theta) = Dec_k(cipher_1) + Dec_k(cipher_2) - Dec_k(cipher_3) \quad (2.9)$$

• Data restoration technique 2:

Assume that the aggregator node $Q_z a^t$ from the quarter Q_z for the time slot $[\tau_x, \tau_y]$ with $\tau_x \leq \theta \leq \tau_y$ is exhausted. The environmental data represented in $Enc_{pk}(Q_z a_{PH_s}^\theta)$ can be recovered as follows. Firstly, the reader device sends a disaster database query ($Q_z, [\tau_x, \tau_y], +, ttl_{max}, D$) to obtain the environmental data from the remaining aggregator nodes $Q_{z-1} a^t$, $Q_{z+1} a^t$, and $Q_{z+2} a^t$ for a time period $\tau \in [t_x, t_y]$. Secondly, for recovering the data $(Q_z a_{PH_s}^\theta)$, the following is done.

1. The reader device sends a disaster query shown in restoration technique 1 to receive $ciphertext_1 = (Q_{z-1} a_{PH_s}^\theta \oplus Q_z a_{PH_s}^\theta)$.
2. The reader device recovers the information represented in $Q_z a_{PH_s}^\theta$ by subtracting $Dec_k(Q_{z+1} a_{PH_s}^\theta)$ from $ciphertext_1$ after decrypting it.

$$Dec_k(Q_z a_{PH_s}^\theta) = Dec_k(Q_{z-1} a_{PH_s}^\theta \oplus Q_z a_{PH_s}^\theta) - Dec_k(Q_{z+1} a_{PH_s}^\theta) \quad (2.10)$$

Note that these restoration techniques are applicable when only one aggregator node per quarter and per epoch is exhausted. If two or more aggregator nodes from different quarters at the same level exhaust, data is irrevocably lost [23].

3. Constraint analysis

In the previous chapter, the drawbacks stemming from resource restrictions of sensor nodes employed in wireless sensor networks have been introduced. Furthermore, TinyPEDS was introduced, which ensures an energy- and space efficient distributed data storage in wireless sensor networks. The main advantage of this approach stems from the idea of using in-network data aggregation.

In this thesis an asymmetric homomorphic encryption scheme, namely the elliptic curve ElGamal (EC-ElGamal) cryptosystem, is implemented (See Section 2.5.2.2). However, asymmetric cryptosystems are assumed to be very resource demanding. Thus, in the following the constraints of the target platform and the techniques used to overcome these obstacles are presented.

3.1 Implementation platform

The target platform used for the implementation of the EC-ElGamal is the Mica-Z mote [3], depicted in Figure 3.1. The Mica-Z uses the Chipcon CC2420, IEEE 802.15.4 compliant, ZigBee ready radio frequency transceiver (2400 MHz to 2483.5 MHz band) and is powered by 2 AA batteries [4].

According to [4], a Mica-Z mote is equipped with Atmel Atmega128L micro-controller which can be clocked at most at 8 MHz. The architecture of Atmega128L micro-controller ([15]) is based on the RISC architecture. Thus, the data and program code need to be stored separately in memory (RAM) and flash memory (ROM), respectively. The Atmega128L has the following features.

- 128K Bytes of in-system programmable flash memory (ROM)
- 4K Bytes EEPROM
- 4K Bytes internal SRAM
- 32 x 8-bit general purpose working registers and peripheral control registers
- 133 instructions



Figure 3.1: Mica-Z mote, from [3]

- On-chip 2-cycle multiplier
- On-chip debug support via JTAG (IEEE std. 1149.1 compliant) interface
- Two expanded 16-bit timer/counters
- 2.7-5.5V operating voltage
- Dual programmable serial USARTs

Note that the properties listed above do not represent all features of the Atmega128L micro-controller, but only those which are relevant for this implementation. The available memory and power supply imply that the implementation of a public key cryptosystem system on such a platform should begin with analysing possible efficiency techniques in order to decrease the memory requirement and increase the performance of the cryptographic operations. In general, there are two ways to achieve this goal.

- Employing appropriate programming techniques
- Choosing efficient cryptographical algorithms

The detailed analysis of the relevant cryptographical algorithms is studied in Chapter 4. Therefore, the following section is dedicated to analyze programming techniques to employ in this implementation.

3.2 Dealing with constraints by choosing an appropriate programming style

Before deciding on an appropriate programming style, the most important criteria should be determined. That means, it should be decided which resource, namely code performance, code size, or memory requirement, has the highest priority for a wireless sensor network application being implemented. In the case of asynchronous wireless sensor networks, the most important criteria is the overall lifetime that should be as long as possible. Therefore, in the following, the resource restrictions regarding the overall lifetime of a wireless sensor network are analyzed.

The performance of the cryptographic operations is important, because if those operations need too much time to be performed, the nodes performing those operations are exhausted earlier. Accordingly, some parts of a wireless sensor network disappear after a relatively short duration. This reduces the overall lifetime of the wireless sensor network.

The monitored environmental data, which needs to be stored persistently, should be stored in flash memory (ROM). Moreover, the program code is stored in flash memory. Therefore, the code size of the core applications such as encryption or node election protocols should be kept low. If the flash memory space available for the application data, i.e. the sensed values, is limited, the time-period for the data monitoring should be reduced. Otherwise the amount of the monitored data exceed the available storage capacity of the nodes. Note that the time-period for the data monitoring is tied to the overall lifetime of the wireless sensor network, due to the fact that if the time-period is very short, the reader device request the data from a sensor network more frequently via query flooding. Since the energy consumption of query processing and that of corresponding aggregated data response is high, the overall lifetime of the wireless sensor network gets shorter.

Every application being performed on sensor nodes requires memory (RAM) space. Obviously, it can not be assumed that any wireless sensor network application needs to perform only mathematical operations required for encryption schemes, but also other operations required for, e.g. routing, node election protocol, temporary data storage, or query processing. Since parallel processing is not supported in the microcontroller employed in the Mica-Z mote, for any arbitrary time-period only one application is expected being run on sensor nodes.

In conclusion, minimizing the code size, i.e. occupied ROM, turns out to be the most important criteria for the implementation in this thesis. Moreover, the EC-ElGamal should be implemented to run on 4K bytes of memory. The techniques employed for increasing the code performance, reducing the code size, and the memory requirements are introduced in the following subsections.

3.2.1 Minimizing required memory size

Memory space is used to store variables and stack structures. Thus, reducing memory size is mainly achieved by reducing the dependency on global data and stack. Therefore, in order to decrease the stack dependency, the usage of recursive algorithms is avoided in this thesis (See Section 4.3.3.5). For reduction of memory size, the following techniques can be employed in general.

- **Storing global constant variables¹ in EEPROM:** If global constant variables are stored in EEPROM, the usage of memory space is decreased [16]. However, the main disadvantage of this approach is reduced code performance. This is because of the fact that the data read or write from or to the EEPROM is slower than operations from or to RAM [15].
- **Using local variables instead of global variables:** Since global variables are never allocated to registers, the use of global variables generally results

¹Modulus P or elliptic curve base parameters are examples for global constant variables

in extra loads from memory to registers and stores from registers to memory [16]. This means that since micro-controller can not access to memory directly, the data is loaded to general purpose registers before performing the requested operation and the result is stored to memory. Therefore, using global variables not only increases the required memory size, but also decreases the performance of the code due to unnecessary memory reads and loads. Thus, in this implementation, the use of global variables is avoided whenever possible. More precisely, every piece of data, which can be stored in one 8-Bit digit of the micro-controller, is stored in a local variable, while the data requiring multi-digits are stored in local variables.

- **Using dedicated registers:** Since the employed micro-controller has 32 general purpose registers, frequently accessed variables are stored in registers. This kind of programming not only increases the performance, but also reduces the required memory size. By adding the keyword `register` in front of the variable declaration, the compiler is ordered to allocate that variable in a general purpose register if possible [17].
- **Programming in assembler:** In general, with respect to the memory size, the code programmed in assembler is superior to the code programmed in high level programming languages such as C.

3.2.2 Minimizing code size

Minimizing the code size may be achieved with following approach.

- **Using the compiler's optimization features:** This optimization is very dangerous because the optimized code may not work as intended, since compiler may eliminate code like empty loops or adding zeros [57]. For example, if the compiler's flag for code optimization is set, the compiler tends to remove several iterations from the following code piece.

```
for(int i=0;i<100;i++) {}
```

However, this problem can be solved by testing. This means that after the implementation is finished, the optimization flag is set and one checks, whether the code executes as intended.

- **Avoiding usage of standard library routines:** Because standard library routines are written for general purpose, they usually check for many execution cases. Thus, standard library routines generally contain unnecessary code [57]. Therefore, it is better to write dedicated routines fitting the needs, as done in this thesis.
- **Using the smallest applicable data types:** Obviously, it is important to assign variables with exactly needed sizes. This means that for example if the data can be represented in a 8-Bit variable, the use of 16- or 32-Bit variables should be avoided. This may not only significantly reduce the code size, but also increase the code performance, as fewer instructions are needed to compute on smaller data types [16].

- **Programming in assembler:** In general, the code size is smaller when the code is written in assembler instead of using high level languages such as C [45].
- **Other techniques:** Furthermore, as motivated in [16], the following techniques are employed in order to reduce code size.
 - Instead of using `while(expression){}` loops, `do{} while(expression)` loops are used.
 - *post-increment* and *pre-decrement* loop counters and indirect memory addressings are used. Post-increment, denoted with `variable++` means that, e.g. the pointer variables are incremented after access. Similarly, pre-decrement, denoted with `--variable` means that, e.g. the pointer variables are decremented before access.

3.2.3 Increasing code efficiency

The code performance may be increased with the following approaches.

- **Using the compiler's optimization features:** This optimization is not trustable. Thus, it is used in the same way as for minimizing code size described previously [16].
- **Using inline functions:** Inline functions can be declared by putting the keyword `inline` in front of the function definition line. This makes compiler not to call the function, but to copy the function's code in to desired place, whenever it is called. This means that the overhead resulting from a function-call is eliminated and, therefore, the performance of the function is improved. This technique is effective when a function is called frequently and contains very few code lines. However, larger functions tend to fill the available program memory [55]. Since in this thesis the code size is more important than the performance, this technique is not used in the implementation.
- **Loop unrolling:** The highest increase in the performance of the code may be achieved by using so-called loop unrolling. For example, the for-loop

```
for(int i=0;i<5;i++)  
{dummy[i]=0;}
```

is written as

```
dummy[0]=0;  
dummy[1]=0;  
dummy[2]=0;  
dummy[3]=0;  
dummy[4]=0;
```

This reduces loop overheads for index variable maintenance and conditional branch instructions [55]. The main disadvantage of this approach is the increase in code size. Therefore, in this thesis not all the loops are unrolled, but only loops very critical for the overall performance of the application (See Chapter 6).

Table 3.1: Effects of several programming techniques on resource consumptions

Technique	Code size	Code efficiency (speed)	Memory size
Compiler optimization	+, -	+, -	+, -
Standard libraries	-	-	-
Assembler programming	+	+	+
Global variables	+	-	-
Local variables	+	+	+
Inline functions	-	+	0
Loop unrolling	-	+	+
Dedicated registers	+	+	+

- **Using dedicated registers:** If frequently accessed data is stored in registers, no loads from memory is needed. As the CPU unit accesses the registers directly ([15]), the code performs faster.
- **Programming in assembler:** Code written in assembler provides in general the best performance [45]. Since the programmer may use specific instructions making, for example, the carry management easier. Therefore, the code for performance critical operations such as finite field arithmetic operations are coded using assembler in this thesis.

In conclusion, several techniques may be utilized in order to deal with resource constraints of small devices. However, there is always a trade-off between speed, code size, and memory usage. This means that some techniques may reduce code size, i.e. ROM usage, while decreasing code efficiency or increasing memory (RAM) requirements. Therefore, it should be decided, which resource is more important, namely speed, code size, or memory size. As already stated before, code size is the most important criteria in this thesis, while speed and memory size has second and third priority, respectively. Table 3.1 summarizes a subset of techniques studied in the previous subsection. Thereby, - implies that the use of that technique affects the corresponding resource negatively, while positive affects are denoted with +. 0 denotes that the use of that technique has no affect over the resource consumptions. The techniques written in bold are employed in the implementation.

4. Design decisions

4.1 Software architecture

The software architecture of the EC-ElGamal cryptosystem depicted in Figure 4.1 mainly consists of three levels. The lowest level contains finite field arithmetic operations such as *modular addition*, *modular subtraction*, and *modular multiplication*. Since the operations on higher levels are based on the operations on this level, the finite field level is considered to be core level and, therefore, is the most critical level, when code performance is considered. Elliptic curve arithmetic operations are grouped on the second level. The elliptic curve point operations such as *point addition*, *point doubling*, and *scalar point multiplication* represent the arithmetic operations on this level. Finally, on the application level, by using the operations from lower levels any elliptic curve cryptosystem such as EC-ElGamal may be implemented.

The effectiveness of a cryptosystem in terms of memory usage, run-time, and code size is influenced by the employed algorithms. Therefore, it is important to choose the best combination carefully. In order to select the most promising schemes for later implementation, this chapter introduces several algorithms for finite field and elliptic curve level and analyzes them in terms of performance and code size.

4.2 Finite fields

A finite field, also called *Galois Field*, is a field with only a finite number of elements.

The *order* of a finite field is the number of elements in the field. The finite field of the *order* $q = (p^m)$ with odd prime p and $m = 1$ is called *prime field* and the field with the *order* $q = (2^m)$, where m is a positive integer, is called *binary field* [30].

Although finite fields exist for every prime-power order, the prime finite field and the binary finite field are commonly used in cryptography.

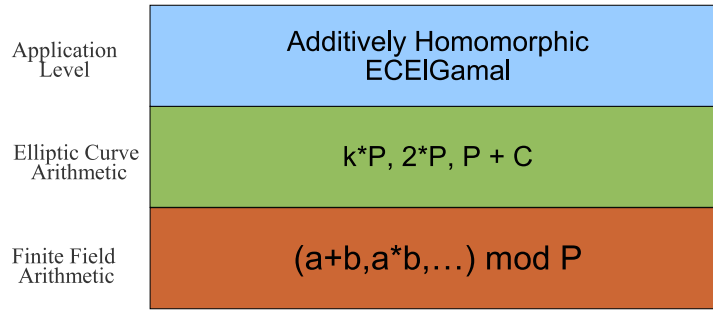


Figure 4.1: Elliptic curve ElGamal (EC-ElGamal) design architecture

4.2.1 Binary field $GF(2^m)$

The binary field denoted $GF(2^m)$ is a vector space of dimension m over the field $GF(2)$. Since the binary field is a vector space, every element a of the field $GF(2^m)$ can be represented uniquely in the form:

$$a = \sum_{m=0}^{m-1} a_m \beta_m \quad (4.1)$$

where $a_m \in 0, 1$ and $\{\beta_0, \beta_1, \dots, \beta_{m-1}\}$ is a *basis* of $GF(2^m)$ over $GF(2)$.

Even though several different *bases* may be used to represent the elements of $GF(2^m)$ over $GF(2)$, polynomial basis representations led to more efficient implementations on micro controller architectures [55]. The following arithmetic operations are defined on elements of $GF(2^m)$ using a polynomial basis representation.

- **Addition:** Bitwise XOR-ing the vector representations.
- **Multiplication:** For two elements of a binary field, $a = (a_{m-1}a_{m-2}\dots a_1a_0)$ and $b = (b_{m-1}b_{m-2}\dots b_1b_0)$, the result of the multiplication $r = a*b = (r_{m-1}r_{m-2}\dots r_1r_0)$ is calculated as follows. First, two operands a and b are multiplied. The result of the multiplication r is computed by reducing the product $a * b$ with a polynomial q , whereby q is an *irreducible polynomial* with degree m .
- **Inversion:** if a is a non-zero element in $GF(2^m)$, the *inverse* of a mod p , denoted a^{-1} , is the unique element $c \in GF(2^m)$ for which $a * c \equiv 1 \pmod{p}$.

4.2.2 Prime field $GF(p)$

The finite field with prime order p is called a prime field and is denoted with $GF(p)$. The prime field is represented by a set of integers modulo p , which are the possible results of the reduction modulo p : $\{0, 1, 2, \dots, p-2, p-1\}$ [30]. The following arithmetic operations are defined on elements of $GF(p)$.

- **Addition:** For $a, b \in GF(p)$, $a + b = r$, where r is the remainder of $a + b$ divided by p , is known as *modular addition* with respect to modulus p .
- **Multiplication:** For $a, b \in GF(p)$, $a * b = s$, where s is the remainder of $a * b$ divided by p is known as *modular multiplication* with respect to modulus p .

- **Inversion:** If a is a non-zero element in $GF(p)$, the *inverse* of $a \bmod p$, denoted a^{-1} , is the unique integer $c \in GF(p)$ for which $a * c \equiv 1 \bmod p$.

4.2.3 Analysis of described finite fields

After the short introduction to possible finite field candidates, this section analyzes them in terms of code size, run-time, and memory performance. Branovic *et al.* studied the memory performance of elliptic curve algorithms in [11]. The authors measured the number of memory accesses and the code sizes of several algorithms over prime and binary fields. In their experiments they found that the number of memory accesses and the code sizes of elliptic curve algorithms for binary fields is higher than those for prime fields. Therefore, on memory constrained devices it is more promising to employ the prime-field arithmetic. Moreover, the binary field arithmetic, particularly multiplication, is not sufficiently supported by usual microprocessors, thus, the use of binary field would lead to lower performance [25].

On the other hand, the main disadvantage of the prime field arithmetic is that the inversion over $GF(p)$ is computationally much more expensive than the inversion over a binary field. However, by using different coordinate systems, the number of inversions required by finite field and elliptic curve arithmetics can be reduced to only one, which is needed for converting the final results back to affine coordinates. Since the cryptosystem to be implemented in this thesis allows those final conversions on external devices such as a laptop, the performance drawback of prime fields stemming from expensive inversions can be ignored.

In contrast to the inversion, *modular reduction* can be performed over prime fields faster than over binary fields when special *Pseudo-Mersenne primes* [62] are used. Therefore, the arithmetic operations in this thesis are implemented over $GF(p)$.

4.3 Finite field arithmetic over $GF(p)$

In this section, several important algorithms for multi-precision arithmetic, i.e. arithmetic with multi-digit operands, are presented. Because the performance of the elliptic curve arithmetics depends heavily on the arithmetic operations described in this section, each candidate algorithm is studied in terms of code size and performance.

Before describing the algorithms in detail, some notation is introduced. Multi-precision integers are unsigned and represented as arrays of w -bit digits. Lowercase letters represents multi-precision integers while the individual w -bit digits are denoted by lowercase letters with index. Due to the computational efficiency, w is chosen as the digit-size of the processor.

For instance, the n -bit integer a is written as

$$a = \sum_{i=0}^{s-1} a_i \cdot 2^{i \cdot w} = a_{s-1} \cdot 2^{(s-1) \cdot w} + \dots + a_1 \cdot 2^w + a_0 \quad (4.2)$$

where s is the number of digits $\lceil n/w \rceil$ and all digits a_i are in the range of $0 \leq a_i < 2^w$.

4.3.1 Multi-precision addition

Addition is performed on two integers having the same number of digits. If the lengths of the integers are different, the smaller integer is first padded with 0's on the left. The multi-precision addition is shown in Algorithm 4.

Algorithm 4 Multi-precision addition [41, p. 594]

Require: two s-digits operands $a = (a_{s-1}a_{s-2}\dots a_1a_0)$, $b = (b_{s-1}b_{s-2}\dots b_1b_0)$

Ensure: $r = a + b = (r_s r_{s-1} \dots r_1 r_0)$

```

1:  $c \leftarrow 0$ 
2: for  $i$  from 0 by 1 to  $s - 1$  do
3:    $r_i \leftarrow (a_i + b_i + c) \bmod 2^w$ 
4:   if  $(a_i + b_i + c) \leq 2^w - 1$  then
5:      $c \leftarrow 0$ 
6:   else
7:      $c \leftarrow 1$ 
8:   end if
9: end for
10:  $r_s \leftarrow c$ 
11: return  $r$ 

```

Since the basic arithmetic operations take place in $GF(p)$, the result of the multi-precision addition has to be reduced. Traditionally, the *multi-precision modular addition* of two integers a and b is performed in the following way. Let p be a modulus. The multi-precision addition is performed by using Algorithm 4 to add a and b . The reduced result z is then calculated as follows:

$$z = \begin{cases} r - p & \text{if } r \geq p \\ r & \text{otherwise} \end{cases}$$

Due to the popular belief that the performance of the addition in cryptosystems affects the overall performance only in a negligible amount, possible improvements of this operation is mostly unattended. This belief is absolutely reliable in case of implementations on powerful devices. However, in sensor networks even small amount of improvements in terms of code size and performance can affect the feasibility of the whole solution. Therefore, in this thesis a *new approach* using the properties of *pseudo-mersenne-primes* for performing modular reduction is proposed in order to save code space and to achieve improvements in performance (see Section 4.3.5.4).

4.3.2 Multi-precision subtraction

The multi-precision subtraction is shown in Algorithm 5.

The reduced result z of the subtraction r is calculated as follows:

$$z = \begin{cases} r & \text{if } a \geq b \\ r + p & \text{otherwise} \end{cases}$$

Algorithm 5 Multiple-Precision subtraction [41, p. 595]

Require: two s -digits perands $a = (a_{s-1}a_{s-2}\dots a_1a_0)$, $b = (b_{s-1}b_{s-2}\dots b_1b_0)$
Ensure: $r = a - b = (r_{s-1}r_{s-2}\dots r_1r_0)$

```

1:  $c \leftarrow 0$ 
2: for  $i$  from 0 by 1 to  $s - 1$  do
3:    $r_i \leftarrow (a_i - b_i + c) \bmod 2^w$ 
4:   if  $(a_i - b_i + c) \geq 0$  then
5:      $c \leftarrow 0$ 
6:   else
7:      $c \leftarrow -1$ 
8:   end if
9: end for
10: return  $r$ 

```

4.3.3 Multi-precision multiplication

The efficiency of the multi-precision multiplication dominates the overall performance. In [25] Gura *et al.* show that on resource constrained devices as much as 85% of execution time is spent on the multi-precision multiplication for a typical point multiplication. That means the optimization of the multi-precision multiplication is critical for overall performance and therefore, it is especially necessary to concentrate on its optimization possibilities in order to increase the performance of the entire cryptosystem.

In this section, five principal methods to perform a multi-precision multiplication namely, Montgomery's method, Schoolbook method, Comba's method, Hybrid method, and Karatsuba's method, are studied.

Because of the limited register space of the constrained devices, multi-precision integers are stored in the memory and loaded digit-wise into the registers when needed. Therefore, multi-precision multiplication not only involves arithmetic operations, but also a significant amount of data transport to and from memory. For example, Schoolbook method, Comba's method, and Hybrid method require exactly s^2 single-precision multiplications for s -digit operands while the number of memory operations (i.e., *load* and *store*) required in each method are different. Therefore, the time requirement of the algorithms depends not only on the arithmetic complexity, but also on the amount of memory operations. This is why in the following the memory performance of each algorithm is analyzed in detail in addition to space and time requirements.

Note that in the following temporary storage denotes the storage in general purpose registers of the employed device, which provides faster reads and writes than reads and writes from memory. For the sake of simplicity, in this thesis the writes and reads from temporary storage are ignored. Additionally, it is assumed that the employed microprocessor can perform subtraction or addition over two operands, whereby each operand is one digit long. Hence, the operation $(u, v) \leftarrow a_j \cdot b_i + z_j + u$ is assumed to require three additions including one additional single-precision addition due to the possible carry-bit in worst case.

4.3.3.1 Montgomery multiplication

Typically, the multiplication of two s -digit integers a and b over $GF(p)$ is performed in two steps. In the first step the integers are multiplied. In the second step the modulo reduction of the $2s$ -digit product from the multiplication is performed, yielding a s -digit result with modulus p , that s digits as well. However, the Montgomery multiplication algorithm [44] merges these two steps in one and is an efficient method for performing modular multiplication.

Given two integers a , b , and the modulus p , the Montgomery multiplication algorithm computes

$$MonMult(a, b) = a \cdot b \cdot r^{-1} \bmod p$$

whereby $a, b < p$ and r is a constant with $gcd(r, p) = 1$.

Even though the algorithm works for any r which is relatively prime to p , it is more useful when r is chosen to be a power of 2. In this case, the Montgomery algorithm performs divisions by a power of 2, which can be achieved by a shift operation. As the shift operation is a relatively fast operation on microprocessors, this leads to a simpler and faster implementation than the implementation of the ordinary modular multiplication [34].

In [34] Koc *et al.* describe a number of efficient software algorithms for calculating the Montgomery product on general purpose processors. All the algorithms proposed by Koc *et al.* require the same number of single-precision multiplication, however, the number of additions and the number of memory accesses are different. According to [34], the *Coarsely Integrated Operand Scanning method (CIOS)* as shown in Algorithm 6 yields better performance than the other methods. Therefore, only the *CIOS* method is analyzed in the following.

The *CIOS* algorithm integrates the multiplication and reduction steps. This means that the *CIOS* method performs the multiplication and reduction steps in the same outer loop, but in different inner loops. This is possible because the value of q in the i th iteration of the outer loop for the reduction depends only on the value z_i , which is completely computed by the i 'th iteration of the outer loop of the multiplication [34]. The advantage of this approach is that only $s + 3$ digits of temporary storage for the intermediate results and 6 digits of temporary storage for keeping operands and constants ($\phi, \beta, \alpha, u, v, \rho$ in Algorithm 6) are needed.

As proposed in [34], the total number of operations can be calculated by counting each operation within a loop, and multiplying this value by the iteration count. Including the final multi-precision subtraction, the *CIOS* method requires $2s^2 + s$ single-precision multiplication, $6s^2 + 3s$ additions, $2s^2 + 2s$ reads and $2s$ writes excluding the writes in temporary storage.

4.3.3.2 Schoolbook multiplication

The Schoolbook multiplication shown in Algorithm 7 is essentially the standard pencil-and-paper multiplication, see [24] and [25]. This algorithm consists of two nested loops, which move through the digits of the operands. In each iteration of

Algorithm 6 Montgomery multiplication (*CIOS*) [34]

Require: two s -digit operands $a = (a_{s-1}a_{s-2}\dots a_1a_0)$, $b = (b_{s-1}b_{s-2}\dots b_1b_0)$, modulus $p = (p_{s-1}p_{s-2}\dots p_1p_0)$, $r = 2^n$ and constant $p'_0 = -p_0^{-1} \bmod 2^w$ with $a, b < p$, $\gcd(p, r) = 1$

Ensure: Montgomery product $z = a \cdot b \cdot r^{-1} \bmod p$

```

1:  $z \leftarrow 0$ 
2:  $\phi \leftarrow p'_0$ 
3: for  $i$  from 0 by 1 to  $s - 1$  do
4:    $u \leftarrow 0$ 
5:    $\beta \leftarrow b_i$ 
6:   for  $j$  from 0 by 1 to  $s - 1$  do
7:      $\alpha \leftarrow a_j$ 
8:      $(u, v) \leftarrow \alpha \cdot \beta + z_j + u$ 
9:      $z_j \leftarrow v$ 
10:  end for
11:   $(u, v) \leftarrow z_s + u$ 
12:   $z_s \leftarrow v$ 
13:   $z_{s+1} \leftarrow u$ 
14:   $u \leftarrow 0$ 
15:   $q \leftarrow z_0 \cdot \phi \bmod 2^w$ 
16:  for  $j$  from 0 by 1 to  $s - 1$  do
17:     $\rho \leftarrow p_j$ 
18:     $(u, v) \leftarrow \rho \cdot q + z_j + u$ 
19:     $z_j \leftarrow v$ 
20:  end for
21:   $(u, v) \leftarrow z_s + u$ 
22:   $z_s \leftarrow v$ 
23:   $z_{s+1} \leftarrow z_{s+1} + u$ 
24:  for  $j$  from 0 by 1 to  $s$  do
25:     $z_j \leftarrow z_{j+1}$ 
26:  end for
27: end for
28: if  $z \geq p$  then
29:    $z \leftarrow z - p$ 
30: end if

```

Algorithm 7 Schoolbook multiplication

Require: two s -digit operands $a = (a_{s-1}a_{s-2}\dots a_1a_0)$, $b = (b_{s-1}b_{s-2}\dots b_1b_0)$ and temporary storage $t = (t_s t_{s-1}\dots t_1 t_0)$

Ensure: product $z = a \cdot b$

```

1:  $t \leftarrow 0$ 
2: for  $i$  from 0 by 1 to  $s - 1$  do
3:    $u \leftarrow b_i$ 
4:   for  $j$  from 0 by 1 to  $s - 1$  do
5:      $v \leftarrow a_j$ 
6:      $t \leftarrow t + u \cdot v \cdot 2^{w \cdot (i+j)}$ 
7:   end for
8:    $z_i \leftarrow t_0$ 
9:   for  $j$  from 0 by 1 to  $s - 1$  do
10:     $t_j \leftarrow t_{j+1}$ 
11:   end for
12:    $t_s \leftarrow 0$ 
13: end for
14: for  $i$  from 0 by 1 to  $s - 1$  do
15:    $z_{i+s} \leftarrow t_i$ 
16: end for

```

the outer loop, a digit b_i of multiplier b is kept constant and multiplied with each single digits of the operand a , before moving to the next multiplier b_{i+1} .

Once the inner loop is finished, the last digit of the integer t , which is employed as temporary storage, contains the last digit of the final result. Therefore, everytime when the execution of the inner loop finished, t_0 is written to the memory and each digits of temporary storage t is shifted one place to the right. In step 14, the s most significant digits of the result are assigned to the final result z .

In this method $s + 1$ digits of temporary storage for the intermediate results are needed. Once the inner loop is completed, the last digit from the temporary storage is stored to memory as a part of the final result. Including the required storage for the digits of the operands in each loop, the Schoolbook method requires $s + 3$ digits of temporary storage.

The number of writes required in this method is equal to the length of the multiplication result, which is $2s$ when the operands are s -digit long. In step 5, the j 'th digit of the operand a is loaded from memory into the temporary storage in each iteration of the inner loop. Since the inner loop iterates s times for each iteration of the outer loop, in total $s^2 + s$ memory accesses are needed to load a_j and b_i from memory to the temporary storage. In addition, the Schoolbook method requires s^2 single-precision multiplications and $2s^2$ additions. The Schoolbook multiplication with 4-digit operands is graphically depicted in Figure 4.2.

4.3.3.3 Comba multiplication

The Comba multiplication, see [24] and [25], is shown in Algorithm 8. The i 'th digit z_i of the product z is the least significant digit of the accumulated partial products $a_j \cdot b_{i-j}$ calculated in the inner loops, whereby $0 \leq j \leq i$. Once the evaluation of the

outer loop is finished, one digit is stored as part of the final multiplication result. The difference to the Schoolbook multiplication is the sequence in which the partial products are calculated, leading to more memory access in the Comba multiplication. Figure 4.2 shows the graphical representation of the Comba multiplication of two multi-precision integers with $s = 2$.

Algorithm 8 Comba multiplication

Require: two s -digit operands $a = (a_{s-1}a_{s-2}\dots a_1a_0)$, $b = (b_{s-1}b_{s-2}\dots b_1b_0)$ and temporary space $t = (t_2t_1t_0)$

Ensure: product $z = a \cdot b$

```

1:  $t \leftarrow 0$ 
2: for  $i$  from 0 by 1 to  $s - 1$  do
3:   for  $j$  from 0 by 1 to  $i$  do
4:      $v \leftarrow a_j$ 
5:      $u \leftarrow b_{i-j}$ 
6:      $t \leftarrow t + u \cdot v \cdot 2^{b \cdot (i+j)}$ 
7:   end for
8:    $z_i \leftarrow t_0$ 
9:    $t_0 \leftarrow t_1$ 
10:   $t_1 \leftarrow t_2$ 
11:   $t_2 \leftarrow 0$ 
12: end for
13: for  $i$  from  $s$  by 1 to  $2s - 2$  do
14:   for  $j$  from  $i - s + 1$  by 1 to  $s - 1$  do
15:      $v \leftarrow a_j$ 
16:      $u \leftarrow b_{i-j}$ 
17:      $t \leftarrow t + u \cdot v \cdot 2^{b \cdot (i+j)}$ 
18:   end for
19:    $z_i \leftarrow t_0$ 
20:    $t_0 \leftarrow t_1$ 
21:    $t_1 \leftarrow t_2$ 
22:    $t_2 \leftarrow 0$ 
23: end for
24:  $z_{2s-1} \leftarrow t_0$ 

```

In each iteration of the inner loop, two digits are multiplied which is added to the accumulative sum of the previous iterations. Since 2-digit long products $u \cdot v$ are added to the cumulative sums from previous loops in steps 6 and 17, the Comba multiplication requires at most three digits temporary storage for keeping the cumulative sum to be used in the next loop.

Each inner loop, shown in Algorithm 8 in steps 3 and 14, is executed $i + 1$ times, where i is the counter of the outer loops. Therefore, each operation of the inner loops is performed

$$\sum_{m=1}^s + \sum_{m=1}^{s-1} = (s \cdot (s + 1))/2 + (s - 1 \cdot (s - 1 + 1))/2 = s^2/2$$

times in total, if the factors have s digits each. There are two memory loads, one single-precision multiplication, and one two-digit addition in each inner loop. Thus, the

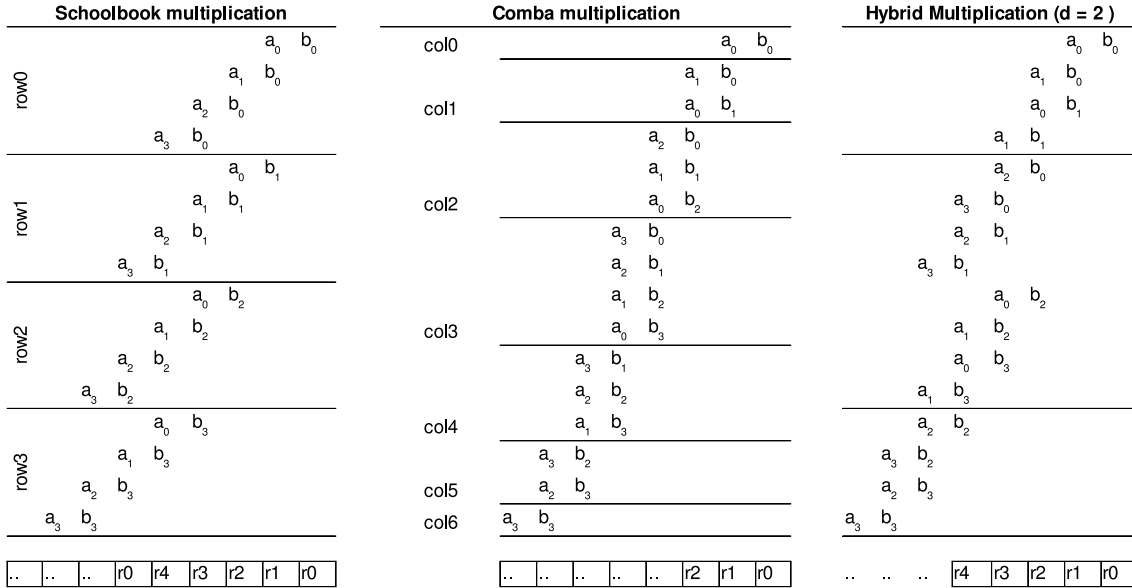


Figure 4.2: Graphical representation of multi-precision multiplication with 4-digit operands

combo multiplication requires $2s^2$ memory loads, s^2 single-precision multiplications, and $2s^2$ single-precision additions. The number of memory writes is the length of the multiplication product, which is $2s$. Because two digits temporary storage are needed to store a_j and b_i in each loop, $3 + 2 = 5$ digits of temporary storage is required to implement the Comba multiplication.

4.3.3.4 Hybrid multiplication

In [25] Gura *et al.* proposed a new multi-precision multiplication algorithm, which is shown in Algorithm 9. This method is essentially the combination of the Schoolbook multiplication and Comba multiplication and inherits the advantages of both algorithms. Specifically, it reduces the need for the temporary storage by using Comba multiplication and the number of memory accesses is decreased as in the Schoolbook multiplication. However, the increased code size is the main drawback of this method.

The two outer nested loops presented in Algorithm 9 stem from the Comba multiplication and the two inner nested loops stem from the Schoolbook multiplication. An example for the hybrid multiplication with 4-digit operands is graphically depicted in Figure 4.2. The optimization of memory accesses is achieved by merging d columns together and then applying the Schoolbook multiplication strategy in each merged column. More specifically, the savings in memory operations stem from the fact that one digit of the multiplier is used in several single-precision multiplications, while loaded from memory only once. This means that b_{j-d+1} is loaded in step 6 from memory only once, while it is used d times in total in the single-precision multiplications presented in step 8.

When column width d is 1, the Hybrid multiplication is equal to the Comba multiplication. The hybrid multiplication equals the Schoolbook method for $d = s$. The

complexity of single-precision multiplication equals the sum of the iterations represented in steps 7 and 21. Because in each of those two loops, one multiplication is performed, yielding s^2 single-precision multiplication. One write operation is needed only for storing the final multiplication result digit-wise to the memory. Thus, $2s$ memory writes are performed while multiplying two s -digit operands.

The memory reads are needed in steps 4, 6, 18, and 20. Steps 4 and 18 are executed $\sum_{m=1}^{\lceil s/d \rceil}$ and $\sum_{m=1}^{\lceil s/d \rceil - 1}$ times, respectively. Therefore, $d \cdot (\lceil s/d \rceil)^2 = \lceil s^2/d \rceil$ memory reads are caused by the operand a . Similarly, the memory reads needed to store single digits of the multiplier b at steps 6 and 20 are equal to $d \cdot (\sum_{m=1}^{\lceil s/d \rceil} + \sum_{m=1}^{\lceil s/d \rceil - 1}) = \lceil s^2/d \rceil$ as well. The length of the accumulated sum of the intermediate results at steps 8 or 22 is at most $2d + \lceil \log_2(s/d)/w \rceil$ long. Therefore, the hybrid method requires $(2d + \lceil \log_2(s/d)/w \rceil + d + 1)$ -digit temporary storage, from which the last two are needed for storing the corresponding digits of a and b , respectively. It is important to note that the temporary storage requirement grows only logarithmically with the increase of the operand size s . The double-precision additions performed in steps 8 and 22 result in $2s^2$ single-precision additions in this method.

4.3.3.5 Karatsuba multiplication

In [31] Karatsuba *et al.* proposed an algorithm with complexity $O(s^{\log_2 3}) \approx O(s^{1.584})$ to compute the product of two s -digit integers. In this method the product z of the two s -digit operands a and b is calculated as follows. First a and b are each divided into two segments of size $s/2$.

$$\begin{aligned} a &= a_h \cdot 2^{s/2} + a_l \\ b &= b_h \cdot 2^{s/2} + b_l \end{aligned}$$

Then the product z can be computed as

$$\begin{aligned} z &= a \cdot b \\ &= (a_h \cdot 2^{s/2} + a_l) \cdot (b_h \cdot 2^{s/2} + b_l) \\ &= a_h \cdot b_h \cdot 2^s + (a_h \cdot b_h + a_l \cdot b_l - (a_h - a_l) \cdot (b_h - b_l)) \cdot 2^{s/2} + a_l \cdot b_l \end{aligned}$$

Thus, the multiplication result can be calculated by three multiplications of size $s/2$ ignoring the additions and subtractions as shown in Figure 4.3. Note that if the number of digits s is odd, a_h is padded with 0's from left. In this method the three half-size multiplications may be performed with Schoolbook method, Comba method, Hybrid method, or even with Karatsuba method again. Since the single-precision multiplication complexity of the methods such as Schoolbook, Comba, or Hybrid with two s -digit operands is s^2 , Karatsuba multiplication requires only $3s^2/4$ single-precision multiplication, when one of those for half-size multiplication is applied. Furthermore, the recursive Karatsuba multiplication requires only $s^{1.584}$ single-precision multiplication. However, the main drawback of the recursive schema is the increased need for temporary storage to store intermediate results and an increased stack structure. Hence, the recursive Karatsuba multiplication is not appropriate for implementations on constrained devices and, therefore, not considered in the following analysis.

As already stated above, Karatsuba multiplication requires only $3s^2/4$ single-precision multiplications. Thus, Karatsuba multiplication may be employed, if the decrease in

Algorithm 9 Hybrid multiplication [25]

Require: two s -digit operands $a = (a_{s-1}a_{s-2}\dots a_1a_0)$, $b = (b_{s-1}b_{s-2}\dots b_1b_0)$, temporary storages $t = (t_{2\cdot d-1}\dots t_1t_0)$, $t^a = (t_{\lceil s/d \rceil \cdot d-1}^a \dots t_1^a t_0^a)$, t^b , and column width d

Ensure: product $z = a \cdot b$

```

1:  $t \leftarrow 0$ 
2: for  $i$  from 0 by 1 to  $\lceil s/d \rceil - 1$  do
3:   for  $j$  from 0 by 1 to  $i$  do
4:      $(t_{d-1}^a \dots t_0^a) \leftarrow (a_{(i-j+1)\cdot d-1} \dots a_{(i-j)\cdot d})$ 
5:     for  $s$  from 0 by 1 to  $d-1$  do
6:        $t^b \leftarrow b_{j\cdot d+s}$ 
7:       for  $y$  from 0 by 1 to  $d-1$  do
8:          $(t_{2d-1} \dots t_0) \leftarrow (t_{2d-1} \dots t_0) + (t_y^a \cdot t^b \cdot 2^{w\cdot(t+s)})$ 
9:       end for
10:    end for
11:  end for
12:   $(z_{(i+1)\cdot d} \dots z_{i\cdot d}) \leftarrow (t_{d-1} \dots t_0)$ 
13:   $t \leftarrow t \ggg d$ 
14:   $(t_{2d-1} \dots t_d) \leftarrow 0$ 
15: end for
16: for  $i$  from  $\lceil s/d \rceil$  by 1 to  $2\lceil s/d \rceil - 2$  do
17:   for  $j$  from  $i - \lceil s/d \rceil + 1$  by 1 to  $\lceil s/d \rceil - 1$  do
18:      $(t_{d-1}^a \dots t_0^a) \leftarrow (a_{(i-j+1)\cdot d-1} \dots a_{(i-j)\cdot d})$ 
19:     for  $s$  from 0 by 1 to  $d-1$  do
20:        $t^b \leftarrow b_{j\cdot d+s}$ 
21:       for  $y$  from 0 by 1 to  $d-1$  do
22:          $(t_{2d-1} \dots t_0) \leftarrow (t_{2d-1} \dots t_0) + (t_y^a \cdot t^b \cdot 2^{w\cdot(t+s)})$ 
23:       end for
24:     end for
25:   end for
26:    $(z_{(i+1)\cdot d} \dots z_{i\cdot d}) \leftarrow (t_{d-1} \dots t_0)$ 
27:    $t \leftarrow t \ggg d$ 
28:    $(t_{2d-1} \dots t_d) \leftarrow 0$ 
29: end for
30:  $(z_{(i+1)\cdot d} \dots z_{i\cdot d}) \leftarrow (t_{d-1} \dots t_0)$ 

```

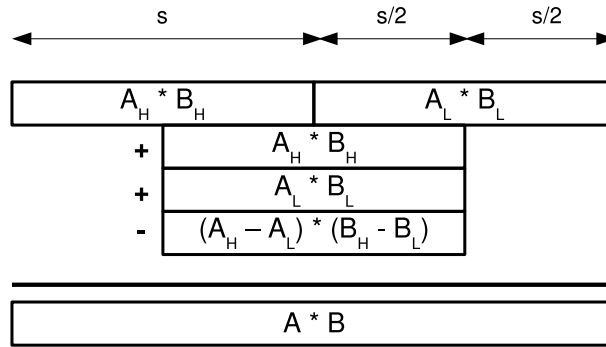


Figure 4.3: Graphical representation of the Karatsuba multiplication

the number of single-precision multiplications is more than the additional overhead stemming from additions and subtractions. Therefore, in the following the computational overhead of the Karatsuba method is analyzed. The possible carry-bits caused by either addition or subtraction are ignored for simplicity.

As the time and space requirements of the half-size multiplications depend merely on the used algorithm, the additional overhead is caused mainly by the additional multi-precision additions and subtractions. As shown in Figure 4.3, in addition to 3 multiplications of size $s/2$ ($a_h \cdot b_h$, $a_l \cdot b_l$, and $(a_h - a_l) \cdot (b_h - b_l)$), the Karatsuba multiplication requires 2 additions and 1 subtraction of size s ($a_h \cdot b_h + a_l \cdot b_l - (a_h - a_l) \cdot (b_h - b_l)$) and 2 subtractions of size $s/2$ ($a_h - a_l$ and $b_h - b_l$). In order to perform 2 subtractions of size $s/2$, $3s$ memory operations and s single-precision additions are required under the assumption that one single-precision subtraction is computationally as expensive as one single-precision addition. Additionally, $5s$ memory operations and $3s$ single-precision additions are performed to calculate 2 additions and 1 subtraction of size s . This means that the additional overhead caused by the Karatsuba method is $10s$ memory operations in total when the computational cost of one memory operation is assumed to be equal to two single-precision additions.

Since the use of the Karatsuba multiplication decreases the number of single-precision multiplications from s^2 to $3s^2/4$, the use of Karatsuba multiplication is only cost-effective with the number of digits s whereby

$$s^2 - 3s^2/4 = s^2/4 \geq 10s$$

In other words, it is not beneficial the use of Karatsuba method if the digit number of the operands are smaller than 40.

4.3.3.6 Analysis of described multiplication algorithms

Table 4.1 summarizes the space requirements, the number of single-precision additions and multiplications, and the memory performance of each algorithm described above.

As already stated, the reads from a memory and writes to a memory are much more expensive than the writes and reads from general purpose registers. For example, on Atmel's Atmega128L microcontroller [15], one read from memory is as expensive

Table 4.1: Storage requirements and memory performance of the multiplication algorithms for s -digit operands

Algorithm	Mult.	Add.	Reads	Writes	Storage
Montgomery (CIOS)	$2s^2 + s$	$6s^2 + 3s$	$2s^2 + 2s$	$2s$	$s + 9$
Schoolbook	s^2	$2s^2$	$s^2 + s$	$2s$	$s + 3$
Comba	s^2	$2s^2$	$2s^2$	$2s$	5
Hybrid	s^2	$2s^2$	$\lceil 2s^2/d \rceil$	$2s$	$3d + 1 + \lceil \log_2(s/d)/w \rceil$

as one single-precision multiplication. Hence, it is possible to increase the efficiency of the multi-precision multiplication by focusing on reducing the writes and reads from memory. This approach is also more advantageous for the implementations on small devices due to the fact that the algorithms aiming to reduce multi-precision multiplications in general need more code space because of the complexity of the algorithm like Karatsuba's recursive method.

However, the problem with using general purpose registers is that the number of the registers available for programming on resource constrained devices are mostly very small. For instance, 8-bit Atmega128L microcontroller has only 32 general purpose registers which can be used as a temporary storage in an implementation.

The evaluation of Table 4.1 shows that the Hybrid method is more promising than the other methods, since it requires not only less temporary storage, but also less number of reads from memory than the other methods. Therefore, for this thesis the Hybrid multiplication algorithm is chosen for the implementation.

4.3.4 Multi-precision squaring

Since the partial products $a_i \cdot a_j$ with $i \neq j$ and $i, j < s$ occurs twice when a s -digit multi-precision integer a is squared, the squaring usually can be performed faster than multi-precision multiplication. For instance, a squaring of two digits integer a requires to compute the partial products $a_1 \cdot a_0$ only once. This is because of the fact that

$$a^2 = a_1^2 \cdot 2^{2w} + 2a_1 \cdot a_0 \cdot 2^w + a_0^2.$$

However, the implementation of the squaring becomes significantly more complex than the multi-precision multiplication while the number of the single-precision multiplications is reduced. This means, there is a trade-off between code size and performance. In Section 6.1, the code size and performance improvements gained by the use of squaring are compared with the performance and code size when multi-precision multiplication for squaring is employed.

4.3.5 Modular reduction

If z is a multi-precision integer, then y , which is the integer remainder in the range $0 \leq y < p$ after z is divided by p , is called the *modular reduction* of z with respect to modulus p [41, p. 599].

After each finite field operation the modular reduction is necessary whenever the result is bigger than the modulus p . Therefore, the performance of the modular reduction is as important as multi-precision multiplication for the overall performance

of the cryptosystem. In the following subsections we analyze three principal methods to perform modular reduction: *Montgomery reduction*, *Barrett reduction*, and the *reduction for pseudo-mersenne primes*.

4.3.5.1 Montgomery reduction

Montgomery reduction was introduced by Peter L. Montgomery in [44] and is an efficient way for performing modular reduction of multi-precision numbers. Although the Montgomery reduction is typically integrated in the Montgomery multiplication to decrease the required memory accesses, it is also possible to employ it alone. The efficiency of this algorithm stems from the idea of not using the computationally expensive division operation. Shift operations, which are computationally cheaper than division, are employed for performing divisions.

Given a multi-precision integer t and modulus p , the Montgomery reduction algorithm calculates

$$\text{MonRed}(t) = t \cdot r^{-1} \bmod p$$

whereby $r > p$, $\gcd(r, p) = 1$, and $0 \leq t < p \cdot r$. If r is chosen to be power of 2, Montgomery reduction can be performed more efficiently.

Contrary to the classical modular reduction $t \bmod p$, the Montgomery reduction calculates $t \cdot r^{-1} \bmod p$. Therefore, the input operand of the Montgomery reduction must be priority converted into the Montgomery representation [41, p. 600] to get rid of the factor r^{-1} . In order to convert a multi-precision integer t in a Montgomery representation, t is multiplied by r . The Montgomery reduction requires $s^2 + s$ single precision multiplications.

4.3.5.2 Barrett reduction

Barrett reduction, proposed by Barrett in [8], is another method to perform modular reduction over multi-precision integers. Barrett reduction is favorable, when used with a single modulus p [41, p. 603]. In order to perform modular reduction, the quantity $\mu = \lfloor 2^{ws}/p \rfloor$ is precomputed. The computation of $t \bmod p$ is shown in Algorithm 10.

Algorithm 10 Barrett reduction [41, p. 604]

Require: multi-precision integer $t = (t_{2s-1}t_{2s-2}\dots t_1t_0)$,

modulus $p = (p_{s-1}p_{s-2}\dots p_1p_0)$ with $p_{s-1} \neq 0$, $b = 2^w$ and $\mu = \lfloor b^{2s}/p \rfloor$

Ensure: $z = t \bmod p$

- 1: $q_1 \leftarrow \lfloor t/b^{(s-1)} \rfloor$, $q_2 \leftarrow q_1 \cdot \mu$, $q_3 \leftarrow \lfloor q_2/b^{s+1} \rfloor$
 - 2: $z_x \leftarrow t \bmod b^{s+1}$, $z_y \leftarrow q_3 \cdot p \bmod b^{s+1}$, $z = z_x - z_y$
 - 3: **if** $z < 0$ **then**
 - 4: $z \leftarrow z + b^{s+1}$
 - 5: **end if**
 - 6: **while** $z \geq p$ **do**
 - 7: $z \leftarrow z - p$
 - 8: **end while**
 - 9: **return** z
-

Similar to the Montgomery reduction, all divisions in this method are performed by simple right-shifts operations. The total number of the single-precision multiplications performed in steps 1 and 2 of the algorithm is $s^2 + 3s + 1$ [41, p. 604].

4.3.5.3 Reduction methods for pseudo-mersenne primes

By taking advantage of primes with a special form, the complexity of the modular reduction can be reduced to a negligible amount [41, p. 605]. The primes specified by SECG [54] belong to the family of such primes. Since our implementation is based on the curves specified by the SECG, the fast reduction techniques may be applied, which are not applicable to general primes.

The n -bit prime $p = 2^n - c$ with $\log_2 c \leq n/2$ is called pseudo-mersenne prime [35, p. 25]. Note that c is the sum of a few powers of two. The efficiency of the modular reduction with modulus $p = 2^n - c$ stems from the fact that

$$2^n - c \equiv 0 \pmod{p} \Leftrightarrow 2^n \equiv c \pmod{p}$$

This means that during the process of modular reduction any term which is multiple of 2^n can be substituted by c , which is much more smaller than 2^n . The substitution with c results in a residue class with smaller bit-length and therefore is the main reason for the efficiency. This behavior is illustrated with an example in the following.

Let a and b be n -bit multi-precision integers and z be the $2n$ -bit product of the multiplication $a \cdot b$. The product z may be represented as

$$z = z_h \cdot 2^n + z_l \equiv z_h \cdot c + z_l \pmod{p} \quad (4.3)$$

where z_h and z_l denotes the n most and least significant bits of the z , respectively. As the bit-length of the c is at most $\lceil n/2 \rceil$, the bit-length of the z after substitution is at most $1.5n$. Since 2^n is substituted by c , which is much more smaller, z strictly decreases for each substitution. Therefore, the fast reduction modulo a pseudo-mersenne prime, shown in Algorithm 11, may be calculated by repeating the substitution a few times and performing a final subtraction of moduli p , i.e. if $z \geq p$.

Algorithm 11 Reduction modulo pseudo-mersenne prime [35, p. 26]

Require: n -bit modulus $p = 2^n - c$ with $\log_2 c \leq \lfloor n/2 \rfloor$, operand $z \geq p$

Ensure: $z \pmod{p}$

- 1: **while** $z \geq 2^n$ **do**
 - 2: $z_l \leftarrow z \pmod{2^n}$
 - 3: $z_h \leftarrow \lfloor z/2^n \rfloor$
 - 4: $z \leftarrow z_h \cdot c + z_l$
 - 5: **end while**
 - 6: **if** $z \geq p$ **then**
 - 7: $z \leftarrow z - p$
 - 8: **end if**
 - 9: **return** z
-

4.3.5.4 Analysis of the described modular reduction algorithms

Assuming that the bit-length of the product t to be reduced is $2n$ bits and c is at most $\lceil n/2 \rceil$ bits long, the multiplication and addition in a while loop in Algorithm 11 executes at most 2 times [41, p. 606]. Specifically, the first iteration of the while loop requires s single-precision multiplications and the further $s/2$ single-precision multiplications are needed in the second iteration at the worst case. This means that Algorithm 11 requires at most $3s/2$ single-precision multiplications for reducing $2s$ -digit integer. Since the value of c is the sum of a few powers of two, these multiplications may be performed by using only shifts. This shows that reduction modulo a pseudo-mersenne prime is superior to the other algorithms introduced above. Note that the ratio of the number of single-precision multiplications requested by Montgomery or Barrett reduction grows with the square of the digit-count, while the number of the single-precision multiplications and shifts required in a reduction modulo pseudo-mersenne prime increase linearly. Hence, the Algorithm 11 is implemented for reducing the result of the multi-precision multiplications and squarings.

4.3.5.5 A new approach for modular multi-precision addition

In this work a new approach is proposed for reducing the result of multi-precision additions over $GF(p)$. The proposed approach improves performance and memory requirement of reduction by taking the advantages of pseudo-mersenne primes. According to our knowledge, such an approach has not been published before.

In the classical reduction method the addition z of two integers are reduced by $z - p$, when z is bigger than the modulus p . However, since the addition z is at most $(s + 1)$ -digit long, the reduction may be performed by adding the c to the first s digits of the result, i.e. z_l , as shown in Equation 4.3. This is because of the fact that the most significant digit, i.e. $s + 1$ 'th digit of the addition result z is always 1 and therefore, its reduction equals to

$$z \bmod p = z_h \cdot 2^n + z_l \equiv z_h \cdot c + z_l \equiv 1 \cdot c + z_l \equiv c + z_l \quad (4.4)$$

This has the advantage that only the digits of the c which are not 0 have to be added to z_l and the number of the memory access is restricted by the number of those non-zero digits and carries happening in adding non-zero digits of c to the z_l . Thus, in the worst case the number of the memory accesses for this approach is equal to about $3n/2$, while the classical reduction method always requires $2n$ memory accesses. Moreover, this approach requires usually very small number of single-precision additions, which is in the worst case equal to s , i.e. when in every single-precision additions one carry-bit occurs. However, the classical reduction method needs to perform always s single-precision subtractions.

Note that this approach does not work for all multi-precision integers due to the fact that the result $c + z_l$ may be bigger than the modulus p for some integers. However, the modular addition in elliptic curve arithmetic is performed over $GF(p)$. Therefore, the multi-precision operands are always smaller than the modulus $p = 2^n - c$. Thus, the proposed method yields always the correct result. This behavior is explained with an example in the following. Let a and b be two integers with the value of $2^n - c - 1$ each, which is the highest possible value. The result of the multi-precision

addition equals $z = 2^{n+1} - 2c - 2 = (2^n - c - 1) + (2^n - c - 1)$. Applying Formula 4.4 yields always the correct reduced result $z \bmod p$ for the addition, since

$$\begin{aligned} z \bmod p &= z_l + c \\ &= \underbrace{2^{n+1} - 2c - 2 - 2^n}_{z_l} + c < \underbrace{2^n - c}_p \\ &= 2^{n+1} - 2c - 2 < 2^{n+1} - 2c \end{aligned}$$

holds for every c and n .

4.4 Elliptic curve arithmetic over $GF(p)$

Elliptic curve arithmetics are based on prime finite field $GF(p)$ within the scope of this work. In the following the mathematical background of elliptic curves over prime field is presented. Moreover, the arithmetic operations defined over elliptic curves such as point doubling, addition, and scalar multiplication are introduced. Subsequently, several algorithms proposed for the implementation of those operations are analyzed in terms of performance and storage requirements.

4.4.1 Introduction to the elliptic curves

Elliptic curves were first introduced as a basis for public key cryptography independently by Koblitz [33] and Miller [42]. As a result of their increased popularity and their commercial acceptance, elliptic curves are standardized by organizations such as ANSI (American National Standards Institute), IEEE (Institute of Electrical and Electronics Engineers), ISO (International Standards Organization), and NIST (National Institute of Standards and Technology) [12]. The security of the elliptic curve cryptosystems stems from the hardness of the well known elliptic curve discrete logarithm problem (ECDLP). Solving ECDLP is much harder than classical discrete logarithm problem defined over a prime finite field. Thus, elliptic curve systems achieve an equivalent level of security with significantly smaller key sizes and memory requirements compared to the other discrete logarithm (DLP) based cryptosystems, e.g. DSA. Therefore, the same level of security may be achieved with less expense of processing power, memory space, band width, and electrical power by using elliptic curve cryptosystems.

An elliptic curve over $GF(p)$ with p elements is defined as the points (x, y) satisfying the curve equation

$$E : y^2 = x^3 + ax + b \bmod p$$

whereby x and $y \in GF(p)$. An elliptic curve forms an additive group, when $a, b \in GF(p)$ is chosen such that the equation

$$4a^3 + 27b^2 \neq 0 \bmod p$$

is satisfied. The curve group over $GF(p)$ consists of the points on the corresponding elliptic curve together with a special point \emptyset , called the point at infinity. There are finitely many points including the neutral element \emptyset on such an elliptic curve which is a desirable property for cryptographic purposes [10].

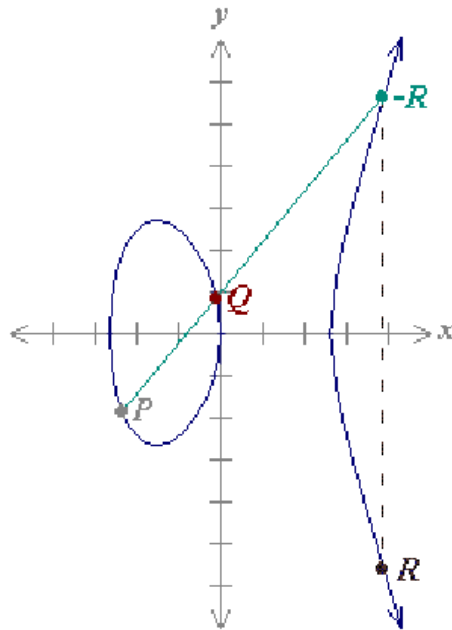


Figure 4.4: Graphical representation of point addition, from [18]

4.4.2 Elliptic curve point addition

Since elliptic curve groups are additive groups, the basic group operation is the point addition. The addition of two points is defined geometrically as depicted in figure 4.4. The reflection of the point $-R$ in the x-axis yields the point R , which is defined to be the result of the addition of two points P and Q . Thereby, the point $-R$ is the intersection point of a straight line drawn through P and Q . The point addition is denoted by ECADD throughout this thesis.

The line drawn through two points P and $-P$ intersects the curve at point infinity \emptyset , therefore, the result of the point addition $P + (-P)$ is defined as \emptyset .

4.4.3 Elliptic curve point doubling

Point doubling adds a point P to itself yielding $R = 2P = P + P$. The point doubling is geometrically depicted in Figure 4.5. In order to double the point P , find the second intersection point $-R$ of a tangent line at the point P with the elliptic curve. The resulting point $R = 2P$ is obtained by mirroring $-R$ at the x-axis.

Note that if the point P lies only in the x-axis, i.e. $y_P = 0$, then the tangent line to the elliptic curve at point P is vertical and does not intersect the elliptic curve at any other point. Hence, the result of the point doubling is defined as \emptyset , when $y_P = 0$. The point doubling is denoted by ECDBL throughout this thesis.

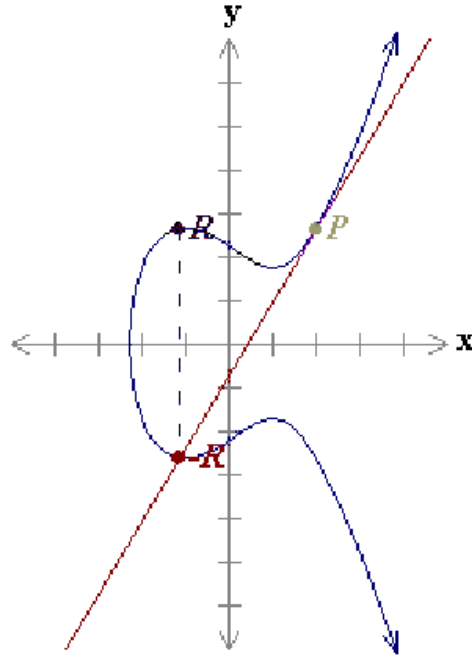


Figure 4.5: Graphical representation of point doubling, from [18]

4.4.4 Selecting the coordinate system

Elliptic curve points may be represented in several coordinate systems. For each of such systems, the speed and the memory requirements of the elliptic curve operations are different. Therefore, a good choice for the coordinate systems is an important factor for the successful implementation of the elliptic curve cryptosystems on resource limited devices. In this section, six popular coordinate systems are discussed in terms of performance and memory requirements namely *Affine*, *Projective*, *Jacobian*, *Chudnovsky-Jacobian*, *modified Jacobian*, and *mixed* coordinate systems in order to decide the best choice for later implementation. The main reference for the coordinate systems is [14].

4.4.4.1 Affine coordinate system

Affine coordinates are the most straight forward. Let E be an elliptic curve over $GF(p)$ given by the equation

$$E : y^2 = x^3 + ax + b \text{ mod } p \quad (4.5)$$

where $a, b \in GF(p)$ and $4a^3 + 27b^2 \neq 0 \text{ mod } p$. Moreover, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points on E with $P \neq \pm Q$. Then the point addition $R = (x_3, y_3) = P + Q$ and the point doubling $R = (x_3, y_3) = P + P$ are defined as follows.

- **Point addition formula in affine coordinates over $GF(p)$:**

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1 \quad (4.6)$$

with $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

- **Point doubling formula in affine coordinates over $GF(p)$:**

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1 \quad (4.7)$$

with $\lambda = (3x_1^2 + a)/(2y_1)$.

For simplicity, in the following time requirements of field multiplications with small constants, e.g. $2x_1$, are ignored. Furthermore, it is assumed that the time requirement of the field addition is equal to that of the field subtraction. Under these assumptions, the computational costs of the ECADD and ECDBL are

$$\begin{aligned} ECADD_a &= 2Mult + 1Inv + 1Sqr + 6Add \\ ECDBL_a &= 2Mult + 1Inv + 2Sqr + 4Add \end{aligned}$$

4.4.4.2 Projective coordinate system

The projective point (x', y', z') is derived from the affine point (x, y) by substituting x and y with x'/z' and y'/z' , respectively. Hence, the projective equation, E_P , of the elliptic curve E is deduced by setting $x = x'/z'$ and $y = y'/z'$ in the equation 4.5, yielding the equation

$$E_p : y'^2 z' = x'^3 + ax' z'^2 + bz'^3 \pmod{p} \quad (4.8)$$

whereby $z' \neq 0$. Let $P = (x'_1, y'_1, z'_1)$ and $Q = (x'_2, y'_2, z'_2)$ be two points on E_p with $P \neq \pm Q$. Then the point addition $R = (x'_3, y'_3, z'_3) = P + Q$ and the point doubling $R = (x'_3, y'_3, z'_3) = P + P$ are defined as follows.

- **Point addition formula in projective coordinates over $GF(p)$:**

$$x'_3 = va', \quad y'_3 = u(v^2 x'_1 z'_2 - a') - v^3 y'_1 z'_2, \quad z'_3 = v^3 z'_1 z'_2 \quad (4.9)$$

with $u = y'_2 z'_1 - y'_1 z'_2$, $v = x'_2 z'_1 - x'_1 z'_2$, $a' = u^2 z'_1 z'_2 - v^3 - 2v^2 x'_1 z'_2$.

- **Point doubling formula in projective coordinates over $GF(p)$:**

$$x'_3 = 2hs, \quad y'_3 = w(4b' - h) - 8y_1'^2 s^2, \quad z'_3 = 8s^3 \quad (4.10)$$

with $w = az_1'^2 + 3x_1'^2$, $s = y_1' z_1'$, $b' = sx_1' y_1'$, $h = w^2 - 8b'$.

The time requirements for additions and doublings in projective coordinates are

$$\begin{aligned} ECADD_p &= 12Mult + 2Sqr + 6Add \\ ECDBL_p &= 7Mult + 5Sqr + 4Add \end{aligned}$$

4.4.4.3 Jacobian coordinate system

The Jacobian point is a triple (x', y', z') derived from the affine point (x, y) by substituting x and y with x'/z'^2 and y'/z'^3 , respectively. Hence, the Jacobian equation, E_j , of the elliptic curve E is deduced by setting $x = x'/z'^2$ and $y = y'/z'^3$ in the equation 4.5, giving the equation

$$E_j : y'^2 = x'^3 + ax'z'^4 + bz'^6 \pmod{p} \quad (4.11)$$

whereby $z' \neq 0$. Let $P = (x'_1, y'_1, z'_1)$ and $Q = (x'_2, y'_2, z'_2)$ be two points on E_j with $P \neq \pm Q$. Then the point addition $R = (x'_3, y'_3, z'_3) = P + Q$ and the point doubling $R = (x'_3, y'_3, z'_3) = P + P$ is defined as follows.

- **Point addition formula in Jacobian coordinates over $GF(p)$:**

$$x'_3 = -h^3 - 2u_1h^2 + r^2, \quad y'_3 = -s_1h^3 + r(u_1h^2 - x'_3), \quad z'_3 = z'_1z'_2h \quad (4.12)$$

with $u_1 = x'_1z'^2_2$, $u_2 = x'_2z'^2_1$, $s_1 = y'_1z'^3_2$, $s_2 = z'^3_1y'_2$, $h = u_2 - u_1$, $r = s_2 - s_1$.

- **Point doubling formula in Jacobian coordinates over $GF(p)$:**

$$x'_3 = t, \quad y'_3 = m(s - t) - 8y'^4_1, \quad z'_3 = 2y'_1z'_1 \quad (4.13)$$

with $s = 4x'_1y'^2_1$, $m = 3x'^2_1 + az'^4_1$, $t = m^2 - 2s$.

The computational costs for ECADD and ECDBL in Jacobian coordinates are

$$\begin{aligned} ECADD_j &= 12Mult + 4Sqr + 6Add \\ ECDBL_j &= 4Mult + 6Sqr + 4Add \end{aligned}$$

4.4.4.4 Chudnovsky-Jacobian coordinate system

In the Chudnovsky-Jacobian coordinate system, an elliptic curve point is represented as a quintuple (x', y', z', z'^2, z'^3) . The additional terms z'^2 and z'^3 aim for faster point additions in this coordinate system.

Let $P = (x'_1, y'_1, z'_1, z'^2_1, z'^3_1)$ and $Q = (x'_2, y'_2, z'_2, z'^2_2, z'^3_2)$ be two points on E_j with $P \neq \pm Q$. Then the point addition $R = (x'_3, y'_3, z'_3, z'^2_3, z'^3_3) = P + Q$ and the point doubling $R = (x'_3, y'_3, z'_3, z'^2_3, z'^3_3) = P + P$ are defined as follows.

- **Point addition formula in Chudnovsky-Jacobian coordinates over $GF(p)$:**

$$x'_3 = -h^3 - 2u_1h^2 + r^2, \quad y'_3 = -s_1h^3 + r(u_1h^2 - x'_3), \quad z'_3 = z'_1z'_2h, \quad z'^2_3 = z'^2_3, \quad z'^3_3 = z'^3_3 \quad (4.14)$$

with $u_1 = x'_1z'^2_2$, $u_2 = x'_2z'^2_1$, $s_1 = y'_1z'^3_2$, $s_2 = z'^3_1y'_2$, $h = u_2 - u_1$, $r = s_2 - s_1$.

- **Point doubling formula in Chudnovsky-Jacobian coordinates over $GF(p)$:**

$$x'_3 = t, y'_3 = m(s - t) - 8y_1^4, z'_3 = 2y_1z'_1, z_3^2 = z_3^2, z_3^3 = z_3^3 \quad (4.15)$$

$$\text{with } s = 4x_1'y_1^2, m = 3x_1'^2 + az_1'^4, t = m^2 - 2s.$$

The computational costs for both operations in Chudnovsky-Jacobian coordinates are

$$ECADD_c = 11Mult + 3Sqr + 6Add$$

$$ECDBL_c = 5Mult + 6Sqr + 4Add$$

4.4.4.5 Modified Jacobian coordinate system

Points in modified Jacobian coordinates are represented as a quadruple (x', y', z', az'^4) . Let $P = (x'_1, y'_1, az_1'^4)$ and $Q = (x'_2, y'_2, az_2'^4)$ be two points on E_{jm} with $P \neq \pm Q$. Then the point addition $R = (x'_3, y'_3, az_3'^4) = P + Q$ and the point doubling $R = (x'_3, y'_3, az_3'^4) = P + P$ are defined as follows.

- **Point addition formula in modified Jacobian coordinates over $GF(p)$:**

$$x'_3 = -h^3 - 2u_1h^2 + r^2, y'_3 = -s_1h^3 + r(u_1h^2 - x'_3), z'_3 = z'_1z'_2h, az_3'^4 = az_3'^4 \quad (4.16)$$

$$\text{with } u_1 = x_1'z_2'^2, u_2 = x_2'z_1'^2, s_1 = y_1'z_2'^3, s_2 = z_1'^3y_2', h = u_2 - u_1, r = s_2 - s_1.$$

- **Point doubling formula in modified Jacobian coordinates over $GF(p)$:**

$$x'_3 = t, y'_3 = m(s - t) - 8y_1^4, z'_3 = 2y_1z'_1, az_3'^4 = 2uaz_1'^4 \quad (4.17)$$

$$\text{with } s = 4x_1'y_1^2, m = 3x_1'^2 + az_1'^4, t = m^2 - 2s, u = 8y_1^4.$$

The computational costs for both operations in modified Jacobian coordinates are

$$(ECADD)_{jm} = 13Mult + 6Sqr + 6Add$$

$$(ECDBL)_{jm} = 4Mult + 4Sqr + 4Add$$

4.4.4.6 Mixed coordinate system

In [14] Cohen *et al.* proposed a mixed coordinate system, which enables faster point additions and doublings. Thus, it is possible to perform point additions and doublings when the points are represented in different coordinate systems. The following notation is used in our analysis in the next section. $C_1C_2C_3$ denotes the point addition where C_1 and C_2 refer to coordinates of the input points and C_3 is the coordinates of the result. Similarly, for point doublings C_1C_2 denotes the coordinates of the input and the result points, respectively.

Table 4.2: Computational efficiencies of point additions and doublings

Field Operation	Point Addition					Point Doubling				
	<i>A</i>	<i>P</i>	<i>J</i>	<i>C</i>	<i>M</i>	<i>A</i>	<i>P</i>	<i>J</i>	<i>C</i>	<i>M</i>
Addition	6	6	6	6	6	4	4	4	4	4
Multiplication	2	12	12	11	13	2	7	4	5	4
Squaring	1	2	4	3	6	2	5	6	6	4
Inversion	1	–	–	–	–	1	–	–	–	–
Total [Mult]	33.7	14.7	16.7	14.7	19.7	34.4	12.4	10.4	11.4	8.4

4.4.4.7 Analysis of described coordinate systems

In order to compare the computational costs of the coordinate systems more accurately, some assumptions have to be made on the time requirements of the field operations. According to our implementation of the finite field arithmetic, the ratio between additions and multiplications $Mult/Add$ is ≈ 9 (see Table 6.1). Additionally, it is assumed that the ratio between multiplications and inversions $Inv/Mult$ is 30 as proposed in [14]. The implementation made in this thesis confirms that using the field multiplication is more promising than using a dedicated squaring operation (see Sections 4.3.4 and 6.1). Therefore, squarings are substituted by multiplication operations and the ratio between multiplications and squarings is set to 1. Table 4.2 summarizes the computational efficiencies of the point additions and doublings in different coordinate systems. The points in affine coordinates are represented with two coordinates (x, y) , while three, four, or five coordinates are needed to represent the points in other coordinate systems. Hence, the main advantage of the affine representations is that the memory requirement is smaller than in the other coordinate systems. Additionally, the affine coordinates require the lowest bandwidth. However, the inversions required in point additions and doublings are the major disadvantage of affine coordinates. If Table 4.2 is analyzed carefully, it can be seen that none of the proposed coordinate systems offer the fastest point additions and doublings at the same time. As a solution of this behavior, Cohen *et al.* recommended the idea of using mixed coordinates, which enables faster additions and doublings by inheriting the efficiency of different coordinate systems.

In [29] Hitchcock *et al.* studied several mixed coordinates systems. They found that using *AJM* or *AJJ* coordinates for point additions is the best choice for elliptic curve implementations on the resource constrained devices. Moreover, they recommended *MJ,MM*, or *JJ* coordinates for point doublings. This is why only those coordinate systems are studied in the following.

The algorithms for point additions and doublings in mixed coordinate systems are shown in Algorithm 12 and 13, respectively. Algorithm 12 can perform point additions in *AJM*, *JJM*, *MMM*, *AJJ* and *AMM* coordinates, while point doublings in *MM*, *MJ*, and *JJ* coordinates may be calculated by using Algorithm 13. The results of the analysis of these algorithms in terms of memory requirement and computational performance is represented in Tables 4.3 and 4.4. Note that the values presented in those tables are average values and a is chosen that $a = -3 \pmod{p}$. This reduces the number of finite field multiplications required for point additions by 1 as it can be seen from the step 53 of Algorithm 12, while the number of additions is increased by 1. The number of additions required for point doublings may be

Table 4.3: Computational efficiencies and memory requirements of point additions in mixed coordinates.

Field Operation	Point Addition				
	<i>AJJ</i>	<i>AJM</i>	<i>AMM</i>	<i>JJM</i>	<i>MMM</i>
Addition	10.5	11.5	11.5	11.5	11.5
Multiplication	8	8.5	8.5	12	12
Squaring	4	5	5	6	6
Total [Mult]	13.1	14.7	14.7	19.2	19.2
Total [Temp. storage]	11s	11s	12s	12s	14s

reduced by 1 when $a = -3 \pmod p$ as seen in the step 14 of Algorithm 13. Furthermore, it is assumed that the step 29 in point additions is executed exactly 1 time for every two point additions while the algorithm is never terminated before step 29. Similarly, it is assumed that the step 3 in point doublings is never executed. Because the assumptions made for every coordinate systems are the same, they do not affect the validity of the comparisons.

From Table 4.3 it can be seen that point additions with *AJJ* coordinates are superior to the other mixed coordinate systems. Although, the computational efficiency of the point doublings in *MJ* coordinates are the highest, the point doublings in those coordinates require more temporary storage than the point doublings in *JJ* coordinates as shown in Table 4.4. However, since the results of the point additions in the application level are used as input for the point doublings during the scalar point multiplication, the use of *MJ* coordinates needs one more additional conversion from Jacobian coordinates to modified Jacobian coordinates. This means the computational efficiency of point doublings in *MJ* coordinates is equal to the point doublings in *JJ* coordinates. Thus, *AJJ* and *JJ* coordinates were chosen in the implementation of the point addition and point doubling, respectively.

4.4.5 Scalar point multiplication

The main operation in elliptic curve cryptosystems is the scalar point multiplication. Since the scalar point multiplication is a time critical operation, there are many researches on its efficient implementation in terms of performance and memory requirements. After introducing the mathematical background, several popular methods, which enable an efficient scalar multiplication, are examined to select the most promising ones for the implementation on small devices with limited resources.

Table 4.4: Computational efficiencies and memory requirements of point doublings in mixed coordinates

Field Operation	Point Doubling		
	<i>JJ</i>	<i>MJ</i>	<i>MM</i>
Addition	12	12	13
Multiplication	4	3	4
Squaring	4	4	4
Total [Mult]	9.3	8.3	9.4
Total [Temp. storage]	9s	10s	11s

Algorithm 12 Point addition in mixed coordinates, based on [29]

Require: points $P_1 = (x_1, y_1)$ or (x_1, y_1, z_1) or (x_1, y_1, z_1, az_1^4) and $P_2 = (x_2, y_2, z_2)$ or (x_2, y_2, z_2, az_2^4)

Ensure: $P_3 = (x_3, y_3, z_3)$ or $(x_3, y_3, z_3, az_3^4) = P_1 + P_2$

```

1: if  $P_1 == \emptyset$  then
2:   if doing AJM or AMM then
3:      $az_3^4 \leftarrow az_2^4$ 
4:   end if
5:    $x_3 \leftarrow x_2, y_3 \leftarrow y_2$ 
6:   return  $P_3$ 
7: end if
8: if  $z_2 == 0$  then
9:    $P_3 \leftarrow P_1$ 
10:  return  $P_3$ 
11: end if
12:  $x_3 \leftarrow x_2, y_3 \leftarrow y_2$ 
13: if  $P_1$  is not affine AND  $z_1 \neq 1$  then
14:    $az_3^4 \leftarrow z_1^2$ 
15:    $x_3 \leftarrow x_3 \cdot az_3^4$ 
16:    $az_3^4 \leftarrow z_1 \cdot az_3^4$ 
17:    $y_3 \leftarrow y_3 \cdot az_3^4$ 
18: end if
19:  $az_3^4 \leftarrow z_2^2$ 
20:  $t_1 \leftarrow x_1 \cdot az_3^4$ 
21:  $t_1 \leftarrow t_1 - x_3$ 
22:  $az_3^4 \leftarrow z_2 \cdot az_3^4$ 
23:  $az_3^4 \leftarrow y_1 \cdot az_3^4$ 
24:  $az_3^4 \leftarrow az_3^4 - y_3$ 
25:  $z_3 \leftarrow z_2 \cdot t_1$ 
26: if  $t_1 == 0$  then
27:   if  $az_3^4 == 0$  then
28:      $P_3 \leftarrow P_1$ 
29:     ecdbl( $P_3$ )
30:   else
31:      $z_3 \leftarrow 0$ 
32:      $az_3^4 \leftarrow 0$ 
33:   end if
34:   return  $P_3$ 
35: end if
36: if  $P_1$  is not affine AND  $z_1 \neq 1$  then
37:    $z_3 \leftarrow z_3 \cdot z_1$ 
38: end if
39:  $t_2 \leftarrow t_1^2$ 
40:  $t_1 \leftarrow t_1 \cdot t_2$ 
41:  $y_3 \leftarrow t_1 \cdot y_3$ 
42:  $t_2 \leftarrow x_3 \cdot t_2$ 
43:  $x_3 \leftarrow (az_3^4)^2$ 
44:  $x_3 \leftarrow x_3 - t_1$ 
45:  $x_3 \leftarrow x_3 - t_2$ 
46:  $x_3 \leftarrow x_3 - t_2$ 
47:  $t_2 \leftarrow t_2 - x_3$ 
48:  $t_2 \leftarrow az_3^4 \cdot t_2$ 
49:  $y_3 \leftarrow t_2 - y_3$ 
50: if not doing AJJ then
51:    $az_3^4 \leftarrow z_3^2$ 
52:    $az_3^4 \leftarrow (az_3^4)^2$ 
53:   if  $a == -3 \pmod{P}$  then
54:      $az_3^4 \leftarrow P - 3az_3^4$ 
55:   else
56:      $az_3^4 \leftarrow a \cdot az_3^4$ 
57:   end if
58: end if
59: return  $P_3$ 

```

Algorithm 13 Point doubling in mixed coordinates, based on [29]

Require: point $P_1 = (x_1, y_1, z_1)$ or (x_1, y_1, z_1, az_1^4)

Ensure: $P_2 = (x_2, y_2, z_2)$ or $(x_2, y_2, z_2, az_2^4) = P_1 + P_1$

```

1: if  $z_1 == 0$  then
2:    $x_2 \leftarrow x_1, y_2 \leftarrow y_1$ 
3:   return  $P_2$ 
4: end if
5: if doing JJ then
6:    $t_2 \leftarrow z_1^2$ 
7: end if
8:  $t_1 \leftarrow y_1 + y_1$ 
9:  $z_2 \leftarrow t_1 \cdot z_1$ 
10:  $y_2 \leftarrow y_1^2$ 
11:  $t_1 \leftarrow x_1 + x_1$ 
12:  $t_1 \leftarrow t_1 + t_1$ 
13:  $t_1 \leftarrow t_1 \cdot y_2$ 
14: if doing JJ and  $a == -3 \pmod{P}$  then
15:    $t_3 \leftarrow x_1 - t_2$ 
16:    $x_2 \leftarrow x_1 + t_2$ 
17:    $t_2 \leftarrow x_2 \cdot t_3$ 
18: else
19:    $t_2 \leftarrow x_1^2$ 
20:    $x_2 \leftarrow t_2 + t_2$ 
21:    $t_2 \leftarrow x_2 + t_2$ 
22:    $t_2 \leftarrow t_2 + az_1^4$ 
23: end if
24:  $x_2 \leftarrow t_2^2$ 
25:  $x_2 \leftarrow x_2 - t_1$ 
26:  $x_2 \leftarrow x_2 - t_1$ 
27:  $t_1 \leftarrow t_1 - x_2$ 
28:  $t_2 \leftarrow t_2 \cdot t_1$ 
29:  $y_2 \leftarrow y_1 + y_1$ 
30:  $y_2 \leftarrow y_2^2$ 
31:  $y_2 \leftarrow y_2 + y_2$ 
32: if doing MM then
33:    $t_1 \leftarrow y_2 + y_2$ 
34:    $az_2^4 \leftarrow t_1 \cdot az_1^4$ 
35: end if
36:  $y_2 \leftarrow t_2 - y_2$ 
37: return  $P_2$ 

```

4.4.5.1 Mathematical background

The operation on an elliptic curve E for calculating a point $R = kP \in E$ for a given point $P \in E$ and for an integer k is called scalar point multiplication. The straight forward way to perform a scalar point multiplication is the use of repeated point additions as depicted below.

$$R = kP = \underbrace{P + P + \dots + P + P}_{k \text{ times}} \quad (4.18)$$

However, the performance of this method is unacceptable with respect to the increase in a scalar k . Thus, in the following two more efficient scalar multiplication algorithms are introduced, namely *Left-to-Right* and *Right-to-Left* binary method. They enable faster point multiplications by using chains of point additions and doublings. The reference for binary methods is [19].

4.4.5.2 Left-to-Right binary method

Algorithm 14 Left-to-Right binary method [19, p. 23]

Require: n -bit scalar k in binary representation, Point P on an elliptic curve over $GF(p)$

Ensure: scalar point multiplication $R = kP$

- 1: $R \leftarrow \emptyset$
 - 2: **for** i from $n - 1$ by 1 to 0 **do**
 - 3: $R \leftarrow ECDBL(R)$
 - 4: **if** $k_i == 1$ **then**
 - 5: $R \leftarrow ECADD(R, P)$
 - 6: **end if**
 - 7: **end for**
 - 8: return R
-

In the Left-to-Right method, shown in Algorithm 14, the bits of the binary representation of the scalar k are parsed starting from the most significant non-zero bit. Since the point addition is performed only when the i 'th bit of the scalar k equals 1, which occurs with a probability of $1/2$, the computational efficiency of this method is approximately

$$(n/2) ECADD + n ECDBL$$

operations.

4.4.5.3 Right-to-Left binary method

The Right-to-Left binary method, presented in Algorithm 15, scans the bits of the scalar k starting from the least significant bit. As the point addition is performed when the i 'th bit of the scalar k equals 1, the time request of this method is on average

$$(n/2) ECADD + n ECDBL$$

operations.

Algorithm 15 Right-to-Left binary method [19, p. 21]

Require: n -bit scalar k in binary representation, Point P on an elliptic curve over $GF(p)$ and temporary storage T

Ensure: scalar point multiplication $R = kP$

```

1:  $R \leftarrow \emptyset$ 
2:  $T \leftarrow P$ 
3: for  $i$  from 0 by 1 to  $n - 1$  do
4:   if  $k_i == 1$  then
5:      $R \leftarrow ECADD(R, T)$ 
6:   end if
7:    $T \leftarrow ECDBL(T)$ 
8: end for
9: return  $R$ 

```

4.4.5.4 Analysis of the presented binary methods

Although the computational efficiencies of both binary methods are the same, the memory requirement for the Right-to-Left method is higher. The reason is step 7, in which the Right-to-Left binary method needs to store the result of the point doubling for the next iteration. Furthermore, with the Right-to-Left method it is not possible to perform point additions in mixed coordinates due to the fact that the value of the temporary point T in the Right-to-Left method changes. Consequently, the Left-to-Right binary method is preferred as the basis algorithm for the scalar point multiplication in this thesis.

4.4.6 Speeding up scalar point multiplication

Several techniques may be employed to speed up the scalar point multiplication on elliptic curves. Due to the fact that the scalar multiplication kP can be seen as a chain of repeated point additions and doublings, in general, the scalar multiplication can be speed up by reducing either the number of point additions and doublings at the same time or by reducing only one of them. Thus, in this thesis those techniques are divided into two groups. However, note that some methods from the first group reduce not only the number of the point doublings, but also the number of the point additions and doublings, while only the number of point doublings are reduced by the other methods. The methods from the second group reduce only the number of point additions required to perform the scalar multiplication.

4.4.6.1 Reducing the number of point doublings and additions

4.4.6.1.1 Interleave Method

The *Interleave method* is introduced in [43] and aims at reducing the number of point doublings for *multi-scalar point multiplication*. The speed improvement of this method stems from the simultaneous point doublings performed in step 3 of Algorithm 14. The Interleave method is originally proposed for the multi-scalar multiplications of the form $(k_1 \cdot P_1 + k_2 \cdot P_2 + \dots + k_n \cdot P_n)$ where the scalars and the points (P_1, P_2, \dots, P_n) are priory known and its performance for multi-scalar point multiplication is analyzed in [19]. However, as mentioned in [19, p. 29], the same idea can be used to perform the scalar multiplications of the form kP . This is due

to the fact that the scalar multiplication kP with a n -bit scalar k and a point P may be represented as a sum of t partial multiplications of scalars $(k_t, k_{t-1}, \dots, k_1)$ with base point P as follows

$$k \cdot P = (k_t \cdot 2^{(t-1)n/t} + k_{t-1} \cdot 2^{(t-2)n/t} + \dots + k_2 \cdot 2^{n/t} + k_1) \cdot P = \sum_{i=1}^t k_i \cdot P_i \quad (4.19)$$

whereby each scalar is n/t -bit long and $(P_t, P_{t-1}, \dots, P_1) = (2^{(t-1)n/t} \cdot P, 2^{(t-2)n/t} \cdot P, \dots, 2^{n/t} \cdot P, P)$. Note that if n is not multiple of t , the scalar is padded with 0's to the left.

Since it would be computationally expensive to compute the points P_t, P_{t-1}, \dots, P_2 on the fly, they are typically precomputed and stored. The modified Left-to-Right binary method using this strategy is shown in Algorithm 16, whereby $k_{(j,i)}$ denotes the i 'th bit of the scalar k_j . The scalar point multiplication can be performed with only n/t iterations by using the Interleave method and each iteration in Algorithm 16 requires 1 point doubling. Thus, the total number of point doublings required for scalar multiplication is reduced from n to n/t compared to the Left-to-Right binary method. Since each bit of the scalars is 1 with a probability of $1/2$, the Interleave method on average requires

$$(n/t) \cdot (t \cdot 1/2) \text{ ECADD} + (n/t) \text{ ECDBL} = (n/2) \text{ ECADD} + (n/t) \text{ ECDBL}$$

operations for computing a scalar multiplication $k \cdot P = \sum_{i=1}^t k_i \cdot P_i$. Thereby, the scalars are represented in binary representation and the points $(P_t, P_{t-1}, \dots, P_2)$ are already precomputed. The Interleave method requires the precomputation and storage of $t - 1$ points. Hence, there is a trade-off between memory requirement and performance. In sections 4.4.6.1.3 and 4.4.6.2.4, the memory requirement and performance improvements gained by the Interleave method are discussed.

Algorithm 16 Interleave method

Require: n -bit scalar k in binary representation, Base point $P_1 = P$ and $(t - 1)$ precomputed points $(P_t, P_{t-1}, \dots, P_2)$

Ensure: scalar point multiplication $R = kP$

```

1:  $R \leftarrow \emptyset$ 
2:  $m \leftarrow n/t$ 
3: for  $i$  from  $m - 1$  by 1 to 0 do
4:    $R \leftarrow \text{ECDBL}(R)$ 
5:   for  $j$  from 1 by 1 to  $t$  do
6:     if  $k_{(j,i)} == 1$  then
7:        $R \leftarrow \text{ECADD}(R, P_j)$ 
8:     end if
9:   end for
10: end for
11: return  $R$ 

```

4.4.6.1.2 Shamir method

Another enhancement of the Left-to-Right binary method is the *Lim-Lee method* which is proposed in [37]. The Lim-Lee method aims for reducing the number

Algorithm 17 Shamir method

Require: n -bit scalar k in binary representation, Base point $P_1 = P$ and $(2^t - 2)$ precomputed points $(P_t, P_{t-1}, \dots, P_2)$ and $(b_i \cdot P_1 + b_i \cdot P_2 + \dots + b_i \cdot P_t)$ for $\forall i, i \leq 0 \leq (n/t) - 1$ and $b_i \in \{0, 1\}$

Ensure: scalar point multiplication $R = kP$

- 1: $R \leftarrow \emptyset$
- 2: $m \leftarrow n/t$
- 3: **for** i from $m - 1$ by 1 to 0 **do**
- 4: $R \leftarrow ECDBL(R)$
- 5: **if** $(k_{(1,i)}, k_{(2,i)} \dots k_{(t,i)}) \neq (0, 0, \dots, 0)$ **then**
- 6: $R \leftarrow ECADD(R, k_{(1,i)}P_1 + k_{(2,i)}P_2 + \dots + k_{(t,i)}P_t)$
- 7: **end if**
- 8: **end for**
- 9: return R

of the point doublings and the number of the point additions required for scalar multiplications.

In the original version of the Lim-Lee method, the n -bit scalar k is divided into $t \times v$ subblocks of b -bit, whereby $n = t \cdot v \cdot b$. Therefore, the original Lim-Lee method requires $(2^t - 1)v$ points to precompute and store for calculating kP [37]. In order to reduce the required number of the point precomputations, the original Lim-Lee method can be modified such that the scalar k is divided into t subblocks instead of $t \times v$ subblocks. Note that in [19], such an approach is referred to as *Shamir method*. Similar to [19], in this thesis the modified Lim-Lee method is called the Shamir method as well. The Shamir method, depicted in Algorithm 17, is adjusted for single scalar multiplications of the form kP from the original version for multi-scalar multiplications presented in [19, p. 27]. Remember that $k_{(j,i)}$ denotes the i 'th bit of the scalar k_j .

As in the Interleave method, the n -bit scalar k may be processed as t scalars, since k can be represented as

$$k = (k_t \cdot 2^{(t-1)n/t} + k_{t-1} \cdot 2^{(t-2)n/t} + \dots + k_2 \cdot 2^{n/t} + k_1)$$

Algorithm 17 performs one point doubling operation in each iteration, while one point addition operation is performed when at least one of the corresponding bits of t scalars, i.e. $(k_{(1,i)}, k_{(2,i)} \dots k_{(t,i)})$, is set. Because the corresponding bits of the scalars with a probability of $(1/2^t = \underbrace{(1/2 \cdot 1/2 \cdot \dots \cdot 1/2 \cdot 1/2)}_{t \text{ times}})$ are 0 in the same time, at

least one bit is set with a probability of $(1 - 1/2^t)$. Therefore, the Shamir method on average requires

$$(n/t) \cdot (1 - 1/2^t) ECADD + (n/t) ECDBL$$

operations to compute a scalar multiplication $k \cdot P = \sum_{i=1}^t k_i \cdot P_i$, where the scalars are represented in binary representation and the points $(P_t, P_{t-1}, \dots, P_2)$ and $(b_i \cdot P_1 + b_i \cdot P_2 + \dots + b_i \cdot P_t)$ for $\forall i, i \leq 1 \leq (n/t) - 1$ and $b_i \in \{0, 1\}$ are precomputed. Since the

Table 4.5: Computational efficiencies of Interleave and Shamir method

t	# ECADD		# ECDBL		# Precom.	
	Interleave	Shamir	Interleave	Shamir	Interleave	Shamir
1	$n/2$	$n/2$	n	n	0	0
2	$n/2$	$3n/8$	$n/2$	$n/2$	1	2
3	$n/2$	$7n/24$	$n/3$	$n/3$	2	6
4	$n/2$	$15n/64$	$n/4$	$n/4$	3	14
5	$n/2$	$31n/160$	$n/5$	$n/5$	4	30

number of the points of the form $(P_t, P_{t-1}, \dots, P_2)$ is $t-1$ and 2^t-1-t precomputations are needed to calculate the points of the form $(b_i \cdot P_1 + b_i \cdot P_2 + \dots + b_i \cdot P_t)$, the Shamir method requires the precomputation and storage of $(t-1+2^t-1-t) = 2^t-2$ points in total.

4.4.6.1.3 Analysis of the presented methods

Table 4.5 summarizes the number of precomputed points, point additions, and point doublings required to perform the scalar multiplication for Interleave and Shamir method. Note that t denotes that the scalar k is parsed as multiple-scalar, whereby each scalar k_t is n/t -bit.

If no point is precomputed, the Interleave and Shamir methods do not yield any performance improvements comparing to the Left-to-Right binary method. Additionally, Table 4.5 shows that depending on the number of the precomputed points each method increases the efficiency in a different way. This means that the Shamir method reduces the number of point additions and doublings while the Interleave method reduced only the number of point doublings. Therefore, for a fair comparison, it is necessary to represent the number of point additions and doublings required in each method in a single operation. According to our implementation, the ratio between point additions and doublings A/D is ≈ 1.6 (see Section 6.2.1). Table 4.6 represents the computational costs of the each method in terms point doublings.

It can be easily seen from Table 4.6 that if the target platform is restricted in terms of memory space, meaning that no precomputed point can be stored, the Interleave and Shamir methods do not yield any performance improvement over the Left-to-Right binary method.

However, the Shamir method becomes the most promising way for performing scalar multiplications, when the number of the precomputed points is more than 2. For 1

Table 4.6: The total number of the point doublings required in Interleave and Shamir method

t	# ECDBL		# Precom.	
	Interleave	Shamir	Interleave	Shamir
1	$1.8n$	$1.8n$	0	0
2	$1.3n$	$1.1n$	1	2
3	$1.13n$	$0.79n$	2	6
4	$1.05n$	$0.62n$	3	14
5	$1.0n$	$0.51n$	4	30

precomputed point the Interleave method is superior to the Shamir and Left-to-Right binary method. Note that these results hold only when the scalar k is represented in the binary representation. Due to the limited resources of the target platform only one precomputed point is stored and therefore, the Interleave method is chosen to perform scalar multiplications in this thesis. If the scalars are represented in a signed binary representation, the efficiency of the Interleave method changes. Therefore, in Section 4.4.6.2.4 the analysis of the Interleave method is revisited.

4.4.6.2 Reducing the number of point additions

This section is dedicated to discuss the methods to reduce the number of point additions required for performing scalar multiplications. The algorithms presented in the previous sections imply that the point additions are performed, if the corresponding bit of the scalar is 1. This means that the number of point additions may be further reduced, if the scalar is represented with a low *Hamming weight*, which is the number of the non-zero bits. In general the Hamming weight of a scalar is low, if it is represented in a signed representation. Therefore, the efficiencies of the scalar multiplication methods from previous sections may be further enhanced by using signed representations.

A representation of non-negative integer $d = \sum_i d_i 2^i$, whereby $d_i \in \{0, \pm 1\}$ denotes the signed representation of d . In the following two kinds of signed representations, namely non adjacent form (NAF) and mutual opposite form (MOF) are introduced.

4.4.6.2.1 Non adjacent form (NAF)

NAF is introduced by Reitwiesner in [53]. The most important property of NAF is that no two consecutive bits is non-zero. Every positive integer has a unique NAF presentation which is at most one bit longer than the binary representation of those integers. Moreover, NAF presentation of any integer provides the minimal hamming weight, meaning that the number of non-zero bits is minimal [49]. For example, $15 = (1111)_2$ can be represented as $(1000\bar{1})_2$ by using NAF, where by $\bar{1}$ denotes a negative bit, namely -1 . This means, the scalar multiplication $15P$ would require 4 point additions, when the scalar is represented in binary representation. However, if the scalar 15 is represented in NAF, the scalar multiplication requires only one point addition and one point subtraction. Since the inverse of Jacobian point $Q = (x, y, y)$ is $-Q = (x, -y, z)$ and $-y = (p - y) \bmod p$, one point subtraction requires the same effort as one point addition and one modular subtraction.

The NAF encoding is based on a simple observation: $2^i + 2^{i-1} = 2^{i+1} - 2^{i-1}$. More specifically, this means that every two consecutive bits of the form $(11)_2$ can be substituted by $(10\bar{1})_2$. Therefore, a scalar k represented in binary representation can be converted to NAF by scanning the bits of the scalar k from right-to-left and by replacing the sequence $(11)_2$ with $(10\bar{1})_2$. It is important to note that the carries resulting from the previous sequence replacements have to be added to the binary representation of the scalar. Thus, in NAF methods the bits of the scalars must be scanned from right-to-left and the NAF of the scalars may be one bit longer than their binary representations.

Note that the Hamming weight of the scalar can be further decreased by using larger digit sets. This approach is known as width- w non adjacent form (w NAF).

Algorithm 18 Generation of the w NAF [59]

Require: positive n -bit integer k , and width w

Ensure: w NAF-encoded integer $w\text{NAF}(k)$

```

1:  $i \leftarrow 0$ 
2: while  $k \geq 1$  do
3:   if  $(k)$  is odd then
4:      $k_i \leftarrow 2 - k \bmod 2^w$ 
5:      $k \leftarrow k - k_i$ 
6:   else
7:      $k_i \leftarrow 0$ 
8:   end if
9:    $k \leftarrow k/2$ 
10:   $i \leftarrow i + 1$ 
11: end while
12: return  $(k_n, k_{n-1}, \dots, k_1, k_0)$ 

```

The main drawback of the w NAF representations comparing to the NAF is the increased memory requirements. Note that 2NAF and the NAF are the same.

Algorithm 18 computes the w NAF for positive integers. The average density of non-zero bits for all w NAFs with bit length $n \rightarrow \infty$ is asymptotically $1/(1+w)$ [59].

4.4.6.2.2 Mutual opposite form (MOF)

MOF is proposed by Okeya *et. al.* in [49]. Similar to NAF, each scalar has a unique MOF, which is at most one bit longer than the binary representations. MOF of the binary string d is obtained by performing $2d \ominus d$, whereby \ominus denotes the bitwise subtraction [49, p. 7]. The difference between MOF and NAF is how the integers in a binary representation are encoded. More specifically, MOF encoding is possible by scanning the bits of the integers both from left-to-right or right-to-left. However, because of the carries, right-to-left scanning is necessary for calculating NAF. Thus, MOF representations are more flexible than the NAF representations [49]. Algorithm 19 shows the left-to-right generation of the MOF from the binary representation.

Algorithm 19 Left-to-Right generation of the MOF [49, p. 7]

Require: n -bit integer k in binary representation

Ensure: MOF-encoded integer $(r_n, r_{n-1}, \dots, r_1, r_0) = \text{MOF}(k)$

```

1:  $r_n \leftarrow k_{n-1}$ 
2: for  $i$  from  $n - 1$  by 1 to 1 do
3:    $r_i \leftarrow k_{i-1} - k_i$ 
4: end for
5:  $r_0 \leftarrow -k_0$ 
6: return  $(r_n, r_{n-1}, \dots, r_1, r_0)$ 

```

Analog to NAF, the non-zero density of MOF can be further decreased by applying window methods. The MOF with window width w is denoted as w MOF. The disadvantage of w MOFs is the increased memory requirement. Algorithm 20 may be

Algorithm 20 Left-to-Right generation of the w MOF [49, p. 10]

Require: n -bit integer k in binary representation and width w **Ensure:** w MOF-encoded integer $(r_n, r_{n-1}, \dots, r_1, r_0) = w\text{MOF}(k)$

```

1:  $r_{-1} \leftarrow 0$ 
2:  $r_n \leftarrow 0$ 
3:  $i \leftarrow n$ 
4: while  $i \geq w - 1$  do
5:   if  $k_i = k_{i-1}$  then
6:      $r_i \leftarrow 0$ 
7:      $i \leftarrow i - 1$ 
8:   else
9:      $(r_i, r_{i-1}, \dots, r_{i-w+1}) \leftarrow T_w(k_{i-1} - k_i, k_{i-2} - k_{i-1}, \dots, k_{i-w} - k_{i-w+1})$ 
10:     $i \leftarrow i - w$ 
11:  end if
12: end while
13: if  $i \geq 0$  then
14:    $(r_i, r_{i-1}, \dots, r_0) \leftarrow T_w(k_{i-1} - k_i, k_{i-2} - k_{i-1}, \dots, k_0 - k_1, -k_0)$ 
15: end if
16: return  $(r_n, r_{n-1}, \dots, r_1, r_0)$ 

```

employed for computing w MOF. Note that this algorithm uses left-to-right recording scheme for generating w MOF of the integers. T_w used in the step 9 and 14 in Algorithm 20 denotes the conversions for width w . For instance, the conversions for width $w = 4$ is shown in Table 4.7.

Efficient algorithms for calculating the conversion table for different widths can be found in [49]. The average density of non-zero bits for all w MOFs with bit length $n \rightarrow \infty$ is asymptotically $1/(1+w)$ [49].

Table 4.7: MOF to w MOF mappings for $w = 4$ [49]

$1000 \mapsto 1000$	$\bar{1}\bar{1}10 \mapsto 0030$	$\bar{1}\bar{1}01 \mapsto 0005$	$100\bar{1} \mapsto 0007$
$\bar{1}000 \mapsto \bar{1}000$	$10\bar{1}0 \mapsto 0030$	$\bar{1}\bar{1}\bar{1} \mapsto 0005$	$10\bar{1}\bar{1} \mapsto 0007$
$\bar{1}\bar{1}00 \mapsto 0100$	$\bar{1}\bar{1}\bar{1}0 \mapsto 00\bar{3}0$	$\bar{1}\bar{1}0\bar{1} \mapsto 000\bar{5}$	$\bar{1}001 \mapsto 000\bar{7}$
$\bar{1}\bar{1}00 \mapsto 0\bar{1}00$	$\bar{1}010 \mapsto 00\bar{3}0$	$\bar{1}\bar{1}\bar{1}\bar{1} \mapsto 000\bar{5}$	$\bar{1}01\bar{1} \mapsto 000\bar{7}$

4.4.6.2.3 Analysis of the presented signed representations

This section analyzes the memory requirement, the non-zero density, and the number of the elements, which have to be precomputed and stored, for the described signed representation schemes w MOF and w NAF.

The main difference between w MOF and w NAF is the direction of the recording. Specifically, the w MOF is recorded from left-to-right, while right-to-left recording is performed for the w NAF. This is also the main advantage of w MOF compared to w NAF, when the scalar multiplication is based on Left-to-Right binary method. Thus, if the w NAF is chosen as the signed representation, the whole scalar must be converted in its signed representation before performing the scalar multiplication.

Table 4.8: Comparison of signed representation methods for calculating kP , where k is n -bit long

Scheme	Non-Zero Density	Memory Requirement (bits)	# Precom. Points
w NAF	$1/(1+w)$	$n+1$	$2^{w-2}-1$
w MOF	$1/(1+w)$	w	$2^{w-2}-1$

Therefore, the memory requirement of the w NAF for the scalar k is $n+1$ -bit. However, because the w MOF requires left-to-right scanning of the scalars, the w MOF encoding may be integrated into the scalar multiplication algorithm. Since the scalar multiplication requires to evaluate w -bit of the scalars at once, it is not necessary to encode the scalars completely, but they can be encoded digit by digit. This is why the memory requirement of the w MOF of the scalar k is w -bit and it is very small compared to $n+1$.

According to [49] and [59], the non-zero density of w MOF and w NAF is the same, which is $1/(1+w)$, where w is the width. Moreover, both representations use the same digit-set, namely $\{0, \pm 1, \pm 3, \pm 5, \dots, \pm 2^{w-1}-1\}$ consisting of $2^{w-1}+1$ elements in total [49].

Since the inverse of the point $Q = (x, y)$, which is $-Q = (x, -y)$, can be calculated computationally cheaply as shown in Section 4.4.6.2.1, they can be computed on the fly and do not need to be precomputed and stored. Excluding $\{0, \pm 1\}$, the digit set of the w MOF and w NAF representations has $2^{w-1}+1-3$ elements in total and $(2^{w-1}+1-3)/2 = 2^{w-2}-1$ of them are non-negative. Therefore, the scalar multiplications of the form kP , where k is represented either in the w MOF or w NAF, requires the precomputation and storage of $2^{w-2}-1$ points in total.

Table 4.8 summarizes the memory requirement, the non-zero density and the number of the precomputations for the w MOF and w NAF representations. The memory requirement of the w MOF is significantly smaller than the memory requirement of the w NAF. Therefore, the signed representation used in the following analysis and later implementation is w MOF.

4.4.6.2.4 Re-Analysis of the Interleave method for signed representations

The analysis made in Section 4.4.6.1.3 assumes that the scalars are represented in unsigned binary representation. This section is dedicated to examine the performance improvements of the scalar multiplication, where the scalars are represented in w MOF, compared to the scalar multiplication of the scalars in the binary representation.

According to Section 4.4.6.1.1, the Interleave method requires on average

$$(n/2) ECADD + (n/t) ECDBL$$

operations for computing a scalar multiplication $k \cdot P = \sum_{i=1}^t k_i \cdot P_i$, where the scalars are represented in binary representation and the points $(P_t, P_{t-1}, \dots, P_2)$ are precomputed. As explained in Section 4.4.6.2.2, w MOF representations reduce the

Table 4.9: Costs for the Interleave method for performing point multiplication

w MOF	# ECADD	# ECDBL	# Precom. Points
2MOF	$n/3$	$n/2$	1
3MOF	$n/4$	$n/2$	3
4MOF	$n/5$	$n/2$	7
5MOF	$n/6$	$n/2$	15
6MOF	$n/7$	$n/2$	31

number of point additions required in a scalar multiplication by reducing the Hamming weight. Since the Hamming weight of a scalar k is approximately $1/(1+w)$, when it is represented in w MOF, the Interleave method requires on average

$$(n/t) \cdot (t \cdot 1/(1+w)) \text{ ECADD} + (n/t) \text{ ECDBL} = \\ (n/(1+w)) \text{ ECADD} + (n/t) \text{ ECDBL}$$

operations for computing a scalar multiplication $k \cdot P = \sum_{i=1}^t k_i \cdot P_i$. According to Table 4.8, each scalar needs $2^{w-2} - 1$ precomputed points, therefore, the total number of the points, that have to be precomputed and stored, is

$$(t-1) + t \cdot (2^{w-2} - 1)$$

Note that $(t-1)$ of those precomputations are needed for calculating the points $(P_t, P_{t-1}, \dots, P_2)$ and the signed representation of the scalars $(k_t, k_{t-1}, \dots, k_2)$ require the remaining precomputations, namely $(t \cdot (2^{w-2} - 1))$, whereby $w \geq 2$.

Only one precomputed point, i.e. $t = 2$, is utilized in this thesis to further speed up scalar multiplications of the form $k \cdot P = \sum_{i=1}^t k_i \cdot P_i$. Moreover, Table 4.9 summarizes the required number of the precomputations and computational complexity of the Interleave method in terms of point additions and point doublings for several widths w .

Table 4.9 implies that if the scalars is represented in 2MOF, the computational cost in terms of point additions compared to Table 4.5, i.e. if the scalars is represented in unsigned binary representation, is reduced by $2/3$ and no additional precomputation is needed. Thus, in this thesis the signed representation used is 2MOF (See Section 4.4.6.2) and the scalar multiplication is performed by the Interleave method.

The analysis made in this section and Section 4.4.6.1.3 shows that there is not only one optimal solution to further accelerate the scalar multiplications. The solutions vary depending on the target platform and the applications being implemented. For example, the optimal solution is different when speed is more important than code size or memory consumption.

5. Implementation

The implementation was done on the Mica-Z mote [3], which is a typical device for wireless sensor networks, see Figure 3.1. The operating system employed in the implementation is TinyOS-2.0 [1], an open-source operating system designed for wireless embedded sensor networks.

5.1 Data representation

- **Representation of multi-precision integers:**

A n -digit multi-precision integer a can be represented as

$$a = \sum_{i=0}^{n-1} a_i \cdot 2^{i \cdot w} \quad (5.1)$$

where w is the digit size. Thus, in the software implementation integers are represented by arrays of n elements:

```
FF_DIGIT a[n];
```

Thereby, the type of FF_DIGIT is `uint8_t` denoting an unsigned 8-bit integer.

- **Representation of elliptic curve points:**

Since elliptic curve operations are in mixed coordinates (See Section 4.4.4.7), an elliptic curve point consists of the three coordinates x , y , and z . Thus, an elliptic curve point is represented by the following structure.

```
struct Point
{
    // point's coordinates
    FF_DIGIT x[OPDIGITS];
    FF_DIGIT y[OPDIGITS];
    FF_DIGIT z[OPDIGITS];
};
typedef struct Point Point;
```

whereby OPDIGITS represents the number of the digits which is defined as *macro*.

- **Representation of required elliptic curve paramaters:**

As shown in Section 2.5.2.2, the EC-ElGamal encryption scheme is performed as follows.

$$M = \text{map}(m) \quad (5.2)$$

$$C = \text{enc}(m) = (R, S) = (kG, M + kY) \quad (5.3)$$

whereby C is a cipher, G is the base point of the employed elliptic curve, $Y = xG$, k is a *random* multi-precision integer and x is the secret key. This implies that two parameters, namely G and Y , are required for performing the EC-ElGamal encryption scheme. Therefore, they are calculated offline and represented in the following structure.

```
struct Params
{
    // base point G of the employed elliptic curve
    Point G;
    // precomputed point Y, with Y = xG, where x is a secret key
    Point Y;
};
typedef struct Params Params;
```

- **Representation of signed integers:**

The interleave method, see Section 4.4.6.2.4, employed in the implementation of elliptic curve arithmetic requires integers represented in the signed *MOF* representation. The following structure is used to represent the sign of the bits of multi-precision integers.

```
struct Mof
{
    // one-bit for the bit status
    unsigned bStatus:1;
    // one-bit for the bit sign
    unsigned bSign:1;
};
typedef struct Mof Mof;
```

Note that instead of using `uint8_t`, the structure `Mof` employs *bit fields* in order to keep memory requirement low.

- **Representation of ciphertexts:**

A ciphertext of the EC-ElGamal encryption scheme is represented by the structure:

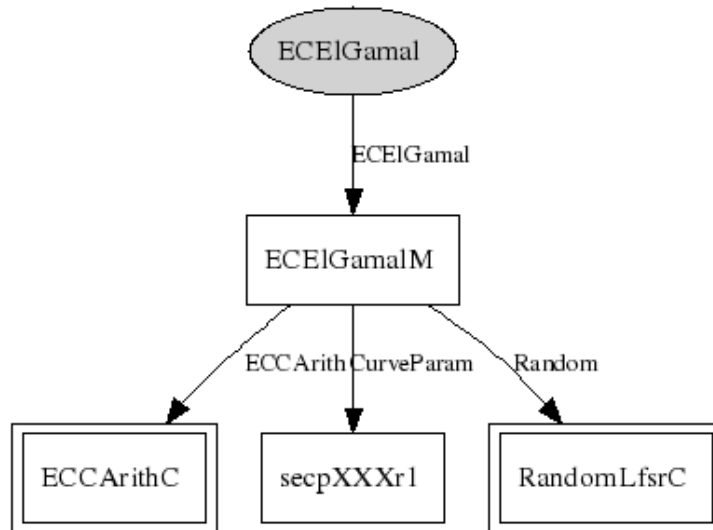


Figure 5.1: Graphical representation of elliptic curve ElGamal implementation

```

struct ECElGamalCipher
{
    // EC-ElGamal cipher (= (R, S))
    Point R;
    Point S;
};
typedef struct ECElGamalCipher ECElGamalCipher;

```

Note that this structure contains two elliptic curve points as an encrypted message has the form of (R, S) .

5.2 Software Components

In TinyOS there are two kinds of components, namely *configurations* and *modules*. Configurations connect modules, while the required functionality, e.g. arithmetic operations, are implemented in modules [36]. Figure 5.1 depicts a graphical representation of the EC-ElGamal configuration. Note that in the following subsections only an important subset of the functions is presented. Functions like help functions are not shown.

5.2.1 ECElGamalM

The module `ECElGamalM` implements the EC-ElGamal encryption scheme and the arithmetic operations such as homomorphic addition operation \boxplus , see Section 2.5.1. Thus, in `ECElGamalM` following functions are implemented.

- `void init()`:
Initializes the parameters, e.g. G, Y and the precomputed points, required by the mapping and the encryption function .

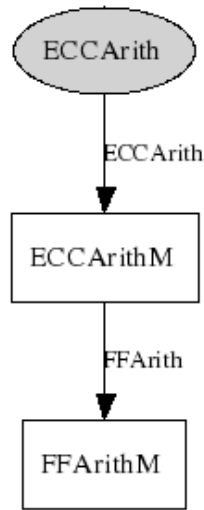


Figure 5.2: Graphical representation of elliptic curve arithmetic implementation

- `void generateRandomNum(FF_DIGIT *k):`

This function generates the random k required in the EC-ElGamal encryption, see equation 5.3. Note that the random number generation is based on the method `rand16()` from the module `RandomLfsrC` which is contained in TinyOS. Therefore, `ECElGama1M` calls the external method `rand16()` and this method call is represented as arrow in Figure 5.1.

- `void map(Point *M, FF_DIGIT *m, FF_DIGIT lengthOfm):`

Software implementation of the mapping function shown in equation 5.2. As described in 2.5.2.2, m is mapped to a elliptic curve point M , whereby $M = mG$.

- `void enc(ECElGamalCipher *cipher, FF_DIGIT *m, FF_DIGIT lengthOfm):`

Software implementation of the EC-ElGamal encryption scheme as described in Section 2.5.2.2, whereby $cipher = enc(m)$.

- `void homAdd(ECElGamalCipher *cipher, ECElGamalCipher *cipher1, ECElGamalCipher *cipher2):`

Software implementation of the homomorphic addition operation (\boxplus), see Section 2.5.1, with $cipher = cipher1 \boxplus cipher2$.

5.2.2 ECCArithC

As depicted in Figure 5.2, the component `ECCArithC` consists of two modules, namely `ECCArithM` and `FFArithM`, which implement the arithmetic operations at elliptic curve and finite field level, respectively.

5.2.2.1 ECCArithM

The module `ECCArithM` implements the following operations from the elliptic curve level.

- `void addAJ(Point *Pres, Point *P1, Point *P2):`
Software implementation of point addition in mixed coordinates, see Algorithm 12. $Pres = P1 + P2$, where `Pres` and `P1` are represented in the jacobian coordinate system, while `P2` is represented in affine coordinate system.
- `void addJJ(Point *Pres, Point *P1, Point *P2):`
Software implementation of point addition in mixed coordinates, see Algorithm 12. That is $Pres = P1 + P2$, where `Pres`, `P1`, and `P2` are represented in the jacobian coordinate system.
- `void double(Point *Pres, Point *P1):`
Software implementation of point doubling in mixed coordinates, see Algorithm 13. That is $Pres = 2P1$, where `Pres` and `P1` are represented in the jacobian coordinate system.
- `void mul(Point *Pres, Point *P1, FF_DIGIT *k, FF_DIGIT lengthOfk):`
Software implementation of scalar point multiplication, whereby $Pres = k \cdot P1$.

5.2.2.2 FFArithM

The module `FFArithM` implements the following finite field arithmetic operations.

- `void modAdd(FF_DIGIT *res, FF_DIGIT *op1, FF_DIGIT *op2):`
Software implementation of modular multi-precision integer addition $res = (op1 + op2) \bmod p$. Note that the modulus p is defined as macro.
- `void modSub(FF_DIGIT *res, FF_DIGIT *op1, FF_DIGIT *op2):`
Software implementation of modular multi-precision integer subtraction. That is $res = (op1 - op2) \bmod p$, where p is defined as macro.
- `void modMult(FF_DIGIT *res, FF_DIGIT *op1, FF_DIGIT *op2):`
Software implementation of modular multiplication with $res = (op1 \cdot op2) \bmod p$.
- `void modSqr(FF_DIGIT *res, FF_DIGIT *op1):`
Software implementation of modular multiplication with $res = op1^2 \bmod p$.

5.2.3 secpXXXr1

Elliptic curve parameters such as the base point G and the point Y and precomputed points are set in this module.

- `void get_param(Params *para):`
The points G and Y are assigned to the corresponding pointers in the structure `para`.
- `void get_PreCompParam(PreCompParams *para, FF_UINT rs):`
The precomputed points required for speeding up point multiplications are assigned to the corresponding pointers in the structure `para`. Note that when $rs == 1$, the precomputed points for kG are requested. Otherwise, the precomputed points for kY are requested.

5.2.4 RandomLfsrC

This module is already implemented in TinyOS and part of the operating system. The following method is employed from this module.

- `command uint16_t rand16():`

Produces a 32-bit pseudorandom number and returns low 16 bits of that random number.

Note that the `RandomLfsrC` does not generate good pseudo-random numbers, which may lead to security problems. However, as they are not within the scope of this thesis, the security analysis of weak pseudo-random numbers is not covered in this work.

6. Implementation results and evaluation

This chapter presents the performance of the algorithms selected in Chapter 4 as the most promising ones. Furthermore, the time requirements of the each operation is compared to existing implementations. Note that the implementation results presented in this chapter are based on the curve *secp160r1* recommended in [54, p.10].

6.1 Implementation results for the finite field arithmetic

6.1.1 Execution time and code size

In order to increase the measurement precision for the operations at finite field level every operation was performed 100 times and the average execution time was taken as result. Table 6.1 shows the time requirements of the finite field arithmetic operations, where execution times are represented in *microseconds* and code sizes in *bytes*. Note that the results for finite field operations presented in this subsection include modular reduction.

As shown in Table 6.1, the finite field arithmetic operations were implemented in three different ways, namely for optimized *code size*, *execution time*, and for *application-optimal* implementation. Note that the application-optimal implementation denotes the trade-off between execution time and code size. In all cases the implementation of subtraction and addition is the same. However, the implementation of squaring and multiplication differs depending on the optimization goal. In case of squaring and multiplication, the code size optimized implementation means that the Schoolbook algorithm (Algorithm 7) was used. The Hybrid multiplication algorithm (Algorithm 9) was employed in the execution time and application-optimal implementation of squaring and multiplication.

For multiplication the difference between the execution time optimized and application-optimal implementation is how the carries are handled. For optimizing the execution

Table 6.1: Performance of the finite field arithmetic operations

160-bit FF Op. (Mod-)	Code size optimized		Execution time optimized		Application-optimal	
	Code size [bytes]	Ex. time [μs]	Code size [bytes]	Ex. time [μs]	Code size [bytes]	Ex. time [μs]
Subt.	152	60	152	60	152	60
Add.	178	41	178	41	178	41
Mult.	358	1000	844	507	640	532
Squa.	358	1000	1088	442	640	532
Total	732	—	2234	—	1008	—

time the carries are added to the intermediate results without checking if the carry flag is set or not. Thus, the additional overheads stemming from checking the carry flag and temporary storage of carries are avoided. Furthermore, the loops performed at step 2 and 16 of Algorithm 9 are unrolled. Therefore, the execution time optimized implementation compared to the application-optimal implementation yields 4.7% better performance, while requiring 24.1% increased code size.

For squaring the difference between the execution time optimized and application-optimal implementation is the employed algorithm. For the execution time optimized version Algorithm 9 was modified such that squaring is accelerated as described in Section 4.3.4. However, squaring was substituted by multiplication for the application-optimal implementation. Therefore, the execution time optimized version compared to the application-optimal implementation is 16.9% faster, while requiring 41.1% more code space.

Supporting the analysis made in Section 4.3.3.6, the application-optimal implementation of the multiplication is about 47% faster than its code size optimized counterpart. However, the code size optimized implementation requires 44% smaller program code space. Moreover, Table 6.1 shows that even though implementing a dedicated squaring operation results in a 16.9% speed improvement, the increase in code size is 41.1%. Since in this work the code size is more important than speed, squaring is substituted by multiplication.

Table 6.2 compares the application-optimal implementation with TinyECC-0.1 [38] and TinyECC-0.2 [39]. In order to measure performance of finite field arithmetic operations TinyECC-0.2 and TinyECC-0.1 were modified such that they could be integrated in our test suite. Note that in the implementation of modular addition, the proposed reduction approach was employed, see Section 4.3.5.5. The experimental results show that compared to TinyECC-0.2, the implementation of modu-

Table 6.2: Performance comparison with TinyECC for finite field arithmetic

160-bit FF Op. (Mod-)	TinyECC-0.1		TinyECC-0.2		This work	
	Code size [bytes]	Ex. time [μs]	Code size [bytes]	Ex. time [μs]	Code size [bytes]	Ex. time [μs]
Subt.	280	91	416	85	152	60
Add.	332	81	456	86	178	41
Mult.	796	2215	1808	701	640	532
Squa.	908	2430	2478	655	640	532
Total	1416	—	3762	—	1008	—

Table 6.3: Summary of the comparison with TinyECC for finite field arithmetic

160-bit (sec160r1)	TinyECC-0.1	TinyECC-0.2
Total code size	28.8% bigger	73.2% bigger
Mod. multiplication	75.8% slower	23.6% slower
Mod. squaring	77.9% slower	18.3% slower

lar addition requires 60% smaller code space, while providing 52% speed improvement. Similarly, the implementation of modular subtraction is superior to that from TinyECC-0.1 and TinyECC-0.2. Table 6.3 summarizes the performance and code size comparison of the squaring and multiplication with both versions of TinyECC.

6.1.2 Power consumption

According to [39], the power consumption P of each arithmetic operation can be calculated by using the formula $P = U \cdot I \cdot t$, whereby U denotes the *voltage*, I denotes the *current*, while the execution time is represented by t . Table 6.4 represents the energy consumption of the finite field arithmetic operations in *nanojoules*. Note that similar to [39], the voltage and the current was assumed to be 3V and 1.8 mA, respectively.

6.2 Implementation results for the elliptic curve arithmetic

In this section the implementation of elliptic curve arithmetic operations, namely point doubling, point addition, and scalar point multiplication, is analyzed. Analog to the performance evaluation of the finite field arithmetic operations, each operation was performed 100 times and the average execution time was taken as result. Note that in this section, the execution times are in *seconds* and the code and memory sizes are in *bytes*.

6.2.1 Execution time and code size

The elliptic curve point doubling implemented by Algorithm 13 requires 4.8 milliseconds, while the elliptic curve point addition (Algorithm 12) is performed in 7.52 milliseconds. The point multiplication operation consists of several point additions and point doublings. Table 6.5 compares the performance, code size, and memory consumption of the point multiplication with existing solutions, namely TinyECC-0.2 [39] and the solution from [25] and [61]. Note that in Table 6.5 the first five

Table 6.4: Power consumption of the finite field arithmetic operations

160-bit (Mod-)	Power consumption
Substraction	324 nJ
Addition	221.4 nJ
Multiplication	2872.8 nJ
Squaring	2872.8 nJ

Table 6.5: Comparison of the scalar point multiplication implementations

160-bit (sec160r1)	#Precom. points	Ex. time [sec.]	Code size [bytes]	Memory size [bytes]
This work	0	1.23	2142	180
This work	0	1.02	2704	421
[25]	0	0.81	3682	282
[61]	0	1.35	n/a	n/a
TinyECC-0.2	0	1.79	6562	382
This work	1	0.68	3160	481
TinyECC-0.2	3	1.76	8444	512
This work	2	0.55	3772	543
TinyECC-0.2	15	2.79	8452	1014
[61]	15	1.24	n/a	n/a

rows imply *general point multiplication*, while the rest of the table represents the results for *fixed point multiplication*. The latter one was further sped up by employing precomputed points.

Execution time, memory usage and code size of the point multiplication from TinyECC-0.2 is not presented in [39]. Thus, the TinyECC-0.2 was integrated in the test suite employed for testing this work. Note that the finite field inversion is required for converting the elliptic curve points from jacobian coordinates to affine coordinates during the decryption process. Since that needs to be performed only on the reader device, the finite field inversion was not implemented. However, since one inversion and four multiplications are needed to covert the Jacobian coordinate to affine at the end of point multiplication, some assumptions are necessary. According to [14], the ratio between finite field multiplications and inversions $Inv/Mult$ is 30. Therefore, for a fair comparison the time requirements of the point multiplication from this work, shown in Table 6.5, represent the actual time requirements + $(34 \cdot 0.532/1000) = 0.018$ sec.

The results presented in the first row of Table 6.5 imply that the point multiplication applies none of the acceleration techniques proposed in Chapter 4, but only the Left-to-Right binary method, see Algorithm 14. The second row represents the results for the scalar point multiplication, whereby integers are represented in 2MOF. For a fair comparison the results are evaluated for fixed and general point multiplication separately.

- **Comparisons for general point multiplication:**

1. **This work vs. [25]:**

In [25] Gura *et al.* implemented the general point multiplication operation. The solution from this work performs a scalar point multiplication 20% slower. Note that the code size and memory consumption for this work, which are shown in Table 6.5, do not include the final conversion from Jacobian to affine. However, the increase in codesize in TinyECC-0.2 was only 234 bytes, when the result of point multiplication was converted to affine coordinate. Therefore, we believe that the code size of this work

will be still about 20% smaller, if the final coordinate conversion is performed. Therefore, the most important criteria for this work, see Chapter 3, is fulfilled.

There are two possible reasons why the solution from [25] is faster, but requires more code space. Firstly, in this work the *loop unrolling* was applied only to loops critical for execution time. However, we believe that Gura et al. applied loop unrolling for every loop for finite field operations and elliptic curve operations. The second possible reason is that according to [25], they implemented everything in assembler. In contrast, in this work only the finite field arithmetic operations were implemented in assembler, while the elliptic curve arithmetic operations were realized in C .

2. This work vs. TinyECC-0.2 [39]:

In [39] Ning *et al.* implemented a point multiplication for general point multiplications. The implementation of this work performs a general point multiplication not only 43% faster, but also its code size 58% smaller. However, the solution from this work results in a slightly increase in memory size, namely 9%.

A possible reason for the increased memory consumption is that this work represents the integers in a signed binary representation (2MOF), while the integers in TinyECC-0.2 are represented by unsigned binary strings. Therefore, the implementation in this work requires some additional memory space for converting the integers into 2MOF.

The increase in the performance for this work stems from the improvements at finite field and elliptic curve level. Firstly, as presented in Table 6.2 the performance of the arithmetic operations at finite field level, on which the performance of the point multiplication is mainly based on, is superior to that of TinyECC-0.2. The improvement at elliptic curve level is achieved by using signed binary representation 2MOF, which enables faster point multiplication.

3. This work vs. [61]:

The implementation from this work compared to [61] performs a general point multiplication 24% faster. Wang *et al.* did not offer any information about code and memory usage of their solution. However, since in their implementation, almost the same acceleration techniques as in TinyECC-0.2 are employed, we assume that the solution of this work is superior to [61] in terms of code size and memory consumption.

• Comparisons for fixed point multiplication:

1. This work vs. TinyECC-0.2 [39]:

The solution from this work performs a fixed point multiplication faster than TinyECC-0.2 and requires smaller code space. As seen from Table 6.5, the implementation of this work is not only 61% faster, but also its code and memory usage is 62% and 6% smaller, respectively. Thereby, the implementation from this work is accelerated by utilizing 1 precomputed point and the TinyECC-0.2 is accelerated by 3 precomputed points. Similarly, when the number of the precomputed points are 2 and 15, the

Table 6.6: Power consumption of the elliptic curve arithmetic operations

#Precom. points (160-bit)	Point mult.	Point add.	Point doubl.
0	5.508 mJ	0.0406 mJ	0.0259 mJ
1	3.672 mJ	0.0406 mJ	0.0259 mJ
2	2.970 mJ	0.0406 mJ	0.0259 mJ

solution of this work is 80% faster and requires 55% and 46% smaller code space and memory space, respectively.

In addition to the reasons stated for general point multiplication, there are several possible reasons which may explain these experimental results. Firstly, TinyECC-0.2 employs the sliding window method to speed up scalar point multiplication operation, while the interleave method (Algorithm 16) accelerates the scalar point multiplication operation in this work. Since the number of the precomputed points required in the sliding window method is a power of 2 [39], it requires to compute and store more precomputed points. In contrast, the interleave method can be performed with fewer number of precomputed points, while yielding higher performance improvement, see Section 4.4.6.1.1. Therefore, it is obvious that the memory usage for the point multiplication in TinyECC-0.2 is higher than in this work. Secondly, in TinyECC-0.2 the points are precomputed on the fly [39], while this work precomputes the points off-line. Therefore, the point multiplication from TinyECC-0.2 requires additional time for point precomputations, which decrease the performance. Note that this is probably the reason why the performance of the point multiplication for TinyECC-0.2 decreases, while the employed number of the precomputed points is increased. As represented in the seventh and ninth row of Table 6.5, according to test suite used in this work the performance of the point multiplication is decreased about 37%, when the number of the precomputed points is increased from 3 to 15. We believe that the reason for this behavior is that the performance improvement gained by more precomputed points may be smaller than the additional computation overhead required for precomputations.

2. This work vs. [61]:

The implementation from this work compared to [61] performs a general point multiplication 56% faster. Wang *et al.* did not offer any information about code and memory consumption of their solution. However, since in their implementation, almost the same acceleration techniques as in TinyECC-0.2 are employed, we assume that the solution of this work is superior to [61] in terms of code size and memory usage.

6.2.2 Power consumption

The power consumption of point doubling, point addition and point multiplication is calculated by the formula introduced in Section 6.1.2. Table 6.6 represents the power consumption of those operations.

Table 6.7: Comparison of the EC-ElGamal encryption scheme implementations

160-bit (sec160r1)	#Precom. points	Ex. time [sec.]	Code size [bytes]	Memory size [bytes]
This work	0	2.52	2790	320
This work	0	2.14	3162	561
TinyECC-0.2	0	3.71	8144	546
This work	2	1.4	3996	621
TinyECC-0.2	3	3.72	10300	676
This work	4	1.17	6158	683
TinyECC-0.2	15	5.62	10308	1178

6.3 Implementation results for the EC-ElGamal encryption scheme

In this section, the implementation of the EC-ElGamal encryption scheme and the homomorphic additive operation \boxplus is analyzed. Analog to the performance evaluations made in previous sections, each operation is performed 100 times and the average execution time is taken as result. Note that in this section, the execution times are in *seconds* and the code and memory sizes are in *bytes*.

6.3.1 Execution time and code size

In order to compare this work with TinyECC-0.2, the EC-ElGamal encryption scheme (Algorithm 1) and the homomorphic additive operation \boxplus (See Section 2.5.1) were implemented in TinyECC-0.2. Table 6.7 compares the code size, memory size, and performance of both implementations. Analog to Table 6.5, the results in the first row imply that the point multiplication employed in the EC-ElGamal is realized by using none of the acceleration techniques proposed in Chapter 4, but only the Left-to-Right binary method (Algorithm 14). The results presented in the second row of the table are obtained by performing the the EC-ElGamal based on the point multiplication, which is accelerated by using 2MOF, see Section 4.4.6.2.2. The results presented in the remainder of the table are obtained by employing the Interleave method. Similarly, the homomorphic addition of two ciphertexts with \boxplus is implemented and tested. The experimental results are depicted in Table 6.8.

Table 6.8: Comparison of the homomorphic addition implementations

160-bit (sec160r1)	#Precom. points	Exec. time [sec.]	Code size [bytes]	Memory size [bytes]
This work	0	2.55	3038	320
This work	0	2.21	3410	561
TinyECC-0.2	0	3.79	8538	546
This work	2	1.46	4264	621
TinyECC-0.2	3	3.77	10718	676
This work	4	1.19	6426	683
TinyECC-0.2	15	5.69	10802	1178

6.3.2 Power consumption

The power consumption of the EC-ElGamal encryption and homomorphic addition is calculated by the formula introduced in Section 6.1.2. Table 6.9 represents the power consumption of the implementations from this work, when those operations are performed.

Table 6.9: Power consumption of the EC-ElGamal encryption and homomorphic addition of two ciphers

#Precom. points (160-bit)	EC-ElGamal encryption	Homomorphic addition
0	11.556 mJ	11.934 mJ
2	7.560 mJ	7.884 mJ
4	6.318 mJ	6.426 mJ

6.4 Summary

The test results presented in previous sections show that the analysis made in Chapter 4 for selecting the most promising algorithms at different levels such as finite field and elliptic curve level is supported by the experimental test results. Moreover, test results imply that the programming style selected in Chapter 3 is appropriate for resource constrained devices. According to our knowledge, in case of a general point multiplication, the implementation from this work is the second fastest implementation after that from [25]. However, our solution's code space requirement is the smallest among the implementations which have been published so far. Moreover, our implementation is the fastest and smallest for a fixed point multiplication.

7. Summary and Conclusions

In this thesis the elliptic curve EC-ElGamal encryption scheme has been implemented and integrated into TinyPEDS for securing the aggregated data storage in asynchronous wireless sensor networks.

Due to the resource restrictions of the sensor nodes, several algorithms required for implementing the EC-ElGamal cryptosystem are analyzed. Thus, the time requirement, code size, and memory consumption of each candidate algorithm were compared and the most promising algorithms were selected and implemented. Moreover, the programming style was selected such that unnecessary overhead in terms of code performance, code size, and memory usage were reduced to minimum.

The analysis for choosing the underlying field, on which the elliptic curve arithmetic based on, showed that the finite field arithmetic over the prime field $GF(p)$ offers better performance for finite field arithmetic operations than the binary field $GF(2^m)$.

Our evaluation for finite field arithmetic operations indicated that the Hybrid multiplication algorithm possesses the highest performance, although it requires slightly more code space. Since the finite field multiplication is the most performance critical operation and its performance affects the overall performance of the EC-ElGamal scheme dramatically, the Hybrid multiplication is employed for the multiplication at finite field level.

The performance of the modular reduction algorithm is the second most important criteria affecting the overall performance of the implemented cryptosystem, due to the fact that each operation over the prime field has to be reduced. The careful study of several candidate reduction algorithms hinted that the reduction algorithm based on pseudo-mersenne primes outperforms other popular reduction algorithms. Moreover, during our analysis we noticed that the reduction of the multi-precision addition over $GF(p)$ can be accelerated by using the same idea. Therefore, we proposed a new approach for reducing the result of the additions at finite field level, which exploits the pseudo-mersenne primes. According to our knowledge, we are the first, who utilized the pseudo-mersenne prime reduction for reducing the addition.

The core operations at elliptic curve level are point addition and point doubling. Since the scalar multiplication, which is the most important operation at this level,

is based on those operations, we examined methods for increasing their performance. Our analysis showed that those elliptic curve operations are performed more efficiently, when they are represented in mixed coordinates.

Additionally, we analyzed several algorithms enabling an efficient scalar point multiplication. Compared to the Right-to-Left binary method, the Left-to-Right binary method is the most promising algorithm for implementing scalar multiplication. This is not only because it requires smaller memory space, but also enables the use of mixed coordinate systems. Furthermore, we applied several techniques to further speed up the scalar point multiplication operation. Firstly, we adapted the Interleave method, which is proposed originally for multi scalar point multiplication, such that it can be employed for a single scalar point multiplication. Secondly, we improved the performance of the Interleave method by representing the scalars in the wMOF representation, which is according to our analysis superior to the more common wNAF representation.

In addition, we implemented all those selected algorithms on Mica-Z mote which is a typical device employed in wireless sensor networks. Our tests showed that scalar point multiplication with a random base takes 1.02 sec, while it takes only 0.55 sec. in the case of fixed point multiplication, when 2 precomputed points are employed. Thus, according to our knowledge, our implementation of scalar point multiplication is the second fastest, but smallest implementation in a general point multiplication. Moreover, our implementation is the fastest and smallest implementation as far as a fixed point multiplication is concerned. These results show that the time requirement for a fixed point multiplication can be reduced to under 0.1 sec, when, e.g., 4 precomputed points for the acceleration are employed. Furthermore, the implementation of the elliptic curve EC-ElGamal encryption showed that the encryption operation takes only 1.17 sec, when in total 4 precomputed points are utilized.

As a conclusion, the experimental results of this work indicate that in contrast to the popular belief, the use of computationally expensive public key encryption schemes are feasible in wireless sensor networks. However, the algorithms required for implementing them should be analyzed, selected and implemented correctly. Moreover, the analysis made in this thesis concludes that there is not only one optimal solution for implementing cryptosystems. An appropriate solution varies depending on the resource restrictions of the implementation platform and the application being implemented.

Future works

The finite field inversion operation is required only for converting the elliptic curve points from jacobian coordinates to affine coordinates during the decryption process. Since that needs to be performed only on the reader device, the finite field inversion was not implemented.

However, our later analysis showed that the costs for transmitting an elliptic curve point in Jacobian coordinates is probably higher than that in affine coordinates. This is because of the fact that Jacobian coordinates are represented by three coordinates, while affine coordinates are represented in two coordinates. In general, the transmission of one coordinate is more expensive than performing one inversion in terms of power consumption. Therefore, we believe that the implementation can

be improved by implementing the finite field inversion over $GF(p)$ and transmitting the data after converting it to affine coordinates.

In addition, under the light of the implementation results, the optimal lifetime of the wireless sensor network network can be estimated. Since the code size of the implementation is known, the remaining memory space available for the persistent storage of the monitored data may be estimated. Using this information, the lifetime of the network is selected such that the monitored data fits the available storage space. Similarly, since the power consumption of the implemented operations are known, therefore, the lifetime of the batteries can be calculated. Obviously, the optimal lifetime of the wireless sensor network is then the smaller one from these two estimations.

Literatur

- [1] <http://www.tinyos.net>.
- [2] *Atmel Corporation.*, March 2007.
<http://www.ortodoxism.ro/datasheets/atmel/2467S.pdf>.
- [3] *Crossbow Technology Inc.*, 2007.
<http://www.xbow.com>.
- [4] *Crossbow Technology Inc.*, 2007.
http://www.xbow.com/Support/Support_pdf_files/MPR-MIB-Series_Users_Manual.pdf.
- [5] *Crossbow Technology Inc.*, 2007.
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.
- [6] J.M. Adler, W. Dai, R.L. Green, and C.A. Neff. Computational details of the votehere homomorphic election system. In *ASIACRYPT*, December 2000.
<http://votehere.net/technicaldocs/hom.pdf>.
- [7] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2004. ACM Press.
<http://almaden.ibm.com/cs/projects/iis/hdb/Publications/papers/opes.pdf>.
- [8] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 311–323, London, UK, 1987. Springer-Verlag.
<http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C86/311.PDF>.
- [9] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, 1994.
<http://citeseer.ist.psu.edu/benaloh94dense.html>.
- [10] Ian F. Blake, Gadiel Seroussi, and Nigel. P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.
- [11] I. Branovic, R. Giorgi, and E. Martinelli. Memory performance of public-key cryptography methods in mobile environments. *ACM SIGARCH Workshop on MEMory performance: DEaling with Applications, systems and architecture (MEDEA-03)*, pages 24–31, 2003.
<http://dii.unisi.it/~giorgi/papers/Branovic03a.pdf>.

- [12] Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. Software implementation of the nist elliptic curves over prime fields. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 250–265, London, UK, 2001. Springer-Verlag.
<http://citeseer.ist.psu.edu/brown01software.html>.
- [13] Claude Castelluccia, Einar Mykletun, and Gene Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. *3rd Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Sensor Networks, Italy*, April 2005.
<http://ics.uci.edu/~gts/paps/mobiq-2005.pdf>.
- [14] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *ASIACRYPT '98: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 51–65, London, UK, 1998. Springer-Verlag.
<http://citeseer.ist.psu.edu/article/cohen98efficient.html>.
- [15] Atmel Corporation. *8-bit Microcontroller with 128K Bytes In-System Programmable Flash*.
<http://atmel.com/atmel/acrobat/doc2467.pdf>.
- [16] Atmel Corporation. *Efficient C Coding for AVR*.
http://atmel-grenoble.com/dyn/resources/prod_documents/doc1497.pdf.
- [17] Atmel Corporation. *How to Program an 8-bit Microcontroller Using C language*.
http://atmel.com/dyn/resources/prod_documents/avr_3_04.pdf.
- [18] Certicom Corporation. *Introduction to ECC*, 2007. ECC Tutorial
<http://certicom.com/index.php?action=ecc,home>.
- [19] Erik Dahmen and Tsuyoshi Takagi. *Efficient Algorithms for Multi-Scalar Multiplications*. Technical University of Darmstadt, Future University–Hakodate, 2005.
<http://cdc.informatik.tu-darmstadt.de/~dahmen/>.
- [20] Danny Dolev and Andrew C. Yao. On the security of public key protocols. Technical report, Stanford, CA, USA, 1981.
- [21] Josep Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, pages 471–483, London, UK, 2002. Springer-Verlag.
<http://vneumann.etse.urv.es/publications/sci/lncs2433.pdf>.
- [22] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
<http://crypto.csail.mit.edu/classes/6.857/papers/elgamal.pdf>.
- [23] Joao Giraio, Dirk Westhoff, Einar Mykletun, and Toshinori Araki. Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks, 2005.
<http://www.ist-ubisecsens.org/publications/Elsevier-2006-GiWeMyAr.pdf>.

- [24] Johann Großschädl, Paolo Ienne, Laura Pozzi, Stefan Tillich, and Ajay K. Verma. Combining algorithm exploration with instruction set design: a case study in elliptic curve cryptography. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 218–223, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. <http://portal.acm.org/citation.cfm?id=1131481.1131543>.
- [25] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *CHES, Boston*, pages 119–132, 2004. <http://research.sun.com/projects/crypto/CHES2004.pdf>.
- [26] Salem Hadim and Nader Mohamed. *Middleware Challenges and Approaches for Wireless Sensor Networks*. IEEE Computer Society, IEEE DS Online Exclusive Content, 2007. <http://computer.org/portal/site/ieeecs/>.
- [27] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks, October 2002. <http://citeseer.ist.psu.edu/heinzelman02applicationspecific.html>.
- [28] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society. <http://citeseer.ist.psu.edu/rabinerheinzelman00energyefficient.html>.
- [29] Yvonne Hitchcock, Edward Dawson, Andrew Clark, and Paul Montague. Implementing an efficient elliptic curve cryptosystem over $gf(p)$ on a smart card. In K. Burrage and Roger B. Sidje, editors, *Proc. of 10th Computational Techniques and Applications Conference CTAC-2001*, volume 44, pages C354–C377, April 2003. <http://citeseer.ist.psu.edu/hitchcock03implementing.html>.
- [30] D. Johnson and A. Menezes. *The Elliptic Curve Digital Signature Algorithm (ECDSA)*. University of Waterloo, 1999. <http://citeseer.ist.psu.edu/johnson99elliptic.html>.
- [31] Anatoly A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963.
- [32] Tetsutaro Kobayashi, Hikaru Morita, Kunio Kobayashi, and Fumitaka Hoshino. Fast elliptic curve algorithm combining frobenius map and table reference to adapt to higher characteristic. In *EUROCRYPT*, pages 176–189, 1999. <http://citeseer.ist.psu.edu/kobayashi99fast.html>.
- [33] Neal Koblitz. *Elliptic curve cryptosystems*. Mathematics of Computation. pp. 203–209, 1987.

- [34] Cetin Kaya Koç, Tolga Acar, and Burton S. Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996. <http://islab.oregonstate.edu/papers/j37acmon.pdf>.
- [35] Sandeep S. Kumar. *Elliptic Curve Cryptography For Constrained Devices*. Ruhr-University Bochum, Bochum, 2006. http://www.crypto.ruhr-uni-bochum.de/en_theses.html.
- [36] Philip Levis. *TinyOS Programming Manual, Revision 1.3*, October 2006. <http://tinysos.net/tinysos-2.x/doc/pdf/tinysos-programming.pdf>.
- [37] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with pre-computation. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 95–107, London, UK, 1994. Springer-Verlag. <http://citeseer.ist.psu.edu/lim94more.html>.
- [38] An Liu and Peng Ning. *TinyECC: Elliptic Curve Cryptography for Sensor Networks*, September 2005. Version 0.1 <http://discovery.csc.ncsu.edu/software/TinyECC/>.
- [39] An Liu and Peng Ning. *TinyECC: Elliptic Curve Cryptography for Sensor Networks*, September 2006. Version 0.2 <http://discovery.csc.ncsu.edu/software/TinyECC/>.
- [40] Wenbo Mao. A structured operational modelling of the dolev-yao threat model. In *Security Protocols Workshop*, pages 34–46, 2002. <http://citeseer.ist.psu.edu/590441.html>.
- [41] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996. <http://www.cacr.math.uwaterloo.ca/hac/>.
- [42] Victor S Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
- [43] Bodo Möller. Algorithms for multi-exponentiation. In *Selected Areas in Cryptography –SAC 2001*, pages 165–180. Springer-Verlag, 2001. <http://cdc.informatik.tu-darmstadt.de/reports/TR/TI-01-08.multiexp.pdf>.
- [44] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.
- [45] Gaute Myklebust. *The AVR Microcontroller and C Compiler Co-Design*, 2007. http://www.atmel.com/dyn/resources/prod_documents/COMPILER.pdf.
- [46] Einar Mykletun, Joao Giraó, and Dirk Westhoff. Public key based cryptoschemes for data concealment in wireless sensor networks. In *IEEE International Conference on Communications*, Istanbul, Turkey, June 2006. ICC2006. <http://citeseer.ist.psu.edu/mykletun06public.html>.

- [47] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 59–66, New York, NY, USA, 1998. ACM Press.
<http://citeseer.ist.psu.edu/naccache98new.html>.
- [48] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *EUROCRYPT*, pages 308–318, 1998.
- [49] Katsuyuki Okeya, Katja Schmidt-Samoa, Christian Spahn, and Tsuyoshi Takagi. Signed binary representations revisited. *CRYPTO 2004*, pages 123–139, August 2004.
<http://citeseer.ist.psu.edu/okeya04signed.html>.
- [50] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.
<http://citeseer.ist.psu.edu/paillier99publickey.html>.
- [51] Pascal Paillier. Trapdooring discrete logarithms on elliptic curves over rings. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 573–584, London, UK, 2000. Springer-Verlag.
<http://citeseer.ist.psu.edu/paillier00trapdooring.html>.
- [52] Min Qin and Roger Zimmermann. An energy-efficient voting-based clustering algorithm for sensor networks. In *SNPD-SAWN '05: Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05)*, pages 444–451, Washington, DC, USA, 2005. IEEE Computer Society.
- [53] George W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.
- [54] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0*, September 2000.
http://secg.org/collateral/sec2_final.pdf.
- [55] Ingo Riedel. *Security in Ad-hoc Networks: Protocols and Elliptic Curve Cryptography on an Embedded Platform*. Ruhr-Universitaet Bochum, NEC Europe Ltd., 2003.
- [56] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On databanks and privacy homomorphisms. *Foundations on Secure Computation*, pages 169–179, 1978.
- [57] ScienceProg. *ARM microcontrollers*, 2007.
<http://www.scienceprog.com/category/arm-microcontrollers/>.
- [58] Michael Sirivianos, Dirk Westhoff, Frederik Armknecht, and Joao Girao. Non-manipulable aggregator node election protocols for wireless sensor networks. In *5th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, Limassol, Cyprus, Greece, April 2007. WiOpt 2007.
<http://www.ist-ubiseconsens.org/publications/sane-fullpaper.pdf>.

-
- [59] Jerome A. Solinas. Efficient arithmetic on koblitz curves. *Des. Codes Cryptography*, 19(2-3):195–249, 2000.
- [60] Tsuyoshi Takagi. *Lecture Slides for Effiziente Kryptographie –SS04*. Technical University of Darmstadt.
http://cdc.informatik.tu-darmstadt.de/TI/Lehre/SS04/Vorlesung/Effiziente_Kryptographie.htm
- [61] Hadong Wang and Qun Li. Efficient Implementation of Public Key Cryptosystems on MICAz and TelosB Motes. Technical report, College of William and Mary, October 2006.
<http://wm.edu/computerscience/techreport/2006/WM-CS-2006-07.pdf>.
- [62] Haodong Wang, Bo Sheng, and Qun Li. TelosB Implementation of Elliptic Curve Cryptography over Primary Field. Technical Report WM-CS-2005-12, College of William and Mary, Computer Science, Williamsburg, VA, October 2005.
<http://cs.wm.edu/~liqun/paper/tr05.pdf>.
- [63] Eric W. Weisstein. Finite field. from MathWorld—A Wolfram Web Resource.
<http://mathworld.wolfram.com/FiniteField.html>.
- [64] Wikipedia. *Wireless Sensor Network*, 2007.
http://en.wikipedia.org/wiki/Sensor_network.
- [65] Ossama Younis and Sonia Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 03(4):366–379, 2004.
<http://citeseer.ist.psu.edu/younis04heed.html>.