12-2007

Self-Certified Public Key Cryptographic Methodologies for Resource-Constrained Wireless Sensor Networks

Ortal Arazi
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Ortal Arazi entitled "Self-Certified Public Key Cryptographic Methodologies for Resource-Constrained Wireless Sensor Networks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Hairong Qi, Major Professor

We have read this dissertation and recommend its acceptance:

Douglas Birdwell, Donald Bouldin, Tom Dunigan

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Ortal Arazi entitled “Self-Certified Public Key Cryptographic Methodologies for Resource-Constrained Wireless Sensor Networks.” I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Hairong Qi, Major Professor

We have read this dissertation
and recommend its acceptance:

Douglas Birdwell

Donald Bouldin

Tom Dunigan

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the
Graduate School

(Original signatures are on file with official student records)

Self-Certified Public Key Cryptographic Methodologies for Resource-Constrained Wireless Sensor Networks

A Dissertation

Presented for the Doctor of Philosophy Degree

Department of Electrical Engineering and Computer Science

The University of Tennessee, Knoxville

Ortal Arazi

December 2007

Dedication

This dissertation is dedicated to my father whose help and patience made it all possible.

Acknowledgments

I would like to thank my advisor, Dr. Hairong Qi, for her guidance and support both academically and personally. I would further like to thank Dr. Donald Bouldin, Dr. J. Douglas Birdwell and Dr. Tom Dunigan, for serving on my Ph.D. committee. The time and input provided is greatly appreciated. This work received considerable benefit from the work of Derek Rose, who assisted in the implementation and evaluation phases presented in this dissertation. Lastly, I would like to thank my family for all of their support. Especially to my father whose insightful remarks helped me throughout all of this process and to my husband for his tremendous encouragement.

Abstract

As sensor networks become one of the key technologies to realize ubiquitous computing, security remains a growing concern. Although a wealth of key-generation methods have been developed during the past few decades, they cannot be directly applied to sensor network environments. The resource-constrained characteristics of sensor nodes, the ad-hoc nature of their deployment, and the vulnerability of wireless media pose a need for unique solutions.

A fundamental requisite for achieving security is the ability to provide for data confidentiality and node authentication. However, the scarce resources of sensor networks have rendered the direct applicability of existing public key cryptography (PKC) methodologies impractical. Elliptic Curve Cryptography (ECC) has emerged as a suitable public key cryptographic foundation for constrained environments, providing strong security for relatively small key sizes.

This work focuses on the clear need for resilient security solutions in wireless sensor networks (WSNs) by introducing efficient PKC methodologies, explicitly designed to accommodate the distinctive attributes of resource-constrained sensor networks. Primary contributions pertain to the introduction of light-weight cryptographic arithmetic operations, and the revision of self-certification (consolidated authentication and key-generation). Moreover, a low-delay group key generation methodology is devised and a denial of service mitigation scheme is introduced. The light-weight cryptographic methods developed pertain to a system-level efficient utilization of the Montgomery procedure and efficient calculations of modular multiplicative inverses. With respect to the latter, computational complexity has been reduced from $O(m)$ to $O(logm)$, with little additional memory cost.

Complementing the theoretical contributions, practical computation off-loading protocols have been developed along with a group key establishment scheme. Implementation on state-of-the-art sensor node platforms has yielded a comprehensive key establishment process obtained in approximately 50 ns, while consuming less than 25 mJ. These exciting results help demonstrate the technology developed and ensure its impact on next-generation sensor networks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Wireless Sensor Networks (WSNs)

Recent advances in wireless network technologies and hardware have yielded multifunctional miniature sensor nodes which are low-cost as well as low-powered. Sensor networks have become one of the key technologies to realize ubiquitous computing, promising to revolutionize our ability to sense and control the physical environment while posing numerous unique challenges to researchers. The vision is to have a sensor network composed of thousands of small wireless nodes efficiently operating autonomously. WSN technology supports a wide range of application domains, including industrial control, home automation and environmental, medical and military monitoring systems. According to a recent National Research Council report [30], WSN technology "could well dwarf previous milestones in information technology".

As a result of cost constraints and the need for ubiquitous, invisible and fast deployments, sensor nodes are physically very small (see figure 1-1 for an illustration), as well as highly resource-constrained [11]. Among the limited resources are energy (units are typically powered by a small battery), processing capabilities, communication range and bandwidth and memory capacity, all of which render the development of wireless nodes a very challenging task.

The three primary functions performed by typical sensor nodes are: (1) to reliably sense and monitor a variety of physical phenomena, (2) to collaborate with other nodes so as to establish an ad-hoc network, and (3) to process, analyze and disseminate the data acquired. Since sensor nodes have limited sensing and computing capabilities, localized collaboration among the nodes

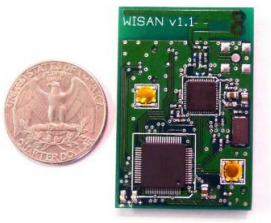

Figure 1-1: Example of a sensor node, emphasizing its small physcial dimensions

is crucial in order to compensate for each other's limitations. Moreover, data correlation is inherently the strongest among nodes that are geographically close to each other. In order to facilitate effective collaboration, the use of a clustering infrastructure is necessary [57]. Such clustering infrastructure should be able to adapt to the dynamically changing environment in which sensor networks operate.

1.2 Unique Security Considerations and Challenges in WSNs

The sensor network, as a network of embedded sensing systems, has been studied extensively since the late 90s. Considerable efforts have been directed towards making them trustworthy [59, 56, 66, 19]. This is particularly true in health and military applications where critical information is frequently exchanged among sensor nodes through insecure wireless media. Traditionally, security is often viewed as a stand-alone component of a system's architecture, for which a dedicated layer is employed. This separation is a flawed approach to network security, particularly in resource-constrained, application-oriented sensor networks. In any application, the security of the system, both in terms of safeguarding against malicious attacks and resilience under malfunction, is a vital component.

Although the area of network security has been studied for decades, the many unique characteristics of sensor networks have traditionally rendered direct application of existing solutions impractical. In particular, the following security considerations and requirements need to be taken into account in the context of sensor networks. *First*, the ad-hoc nature and extreme dynamic environments, in which sensor networks operate, suggest that a prerequisite for achieving security is the ability to encrypt and decrypt confidential data among an *arbitrary* set of sensor

nodes. For the same reason, the keys used for encryption and decryption should be established *at* the nodes instead of using keys generated off-line, prior to deployment. This is important in order to accommodate for the dynamics of the network, as well as the environment. If a communications channel is unavailable during a particular time frame, the protocol should adapt accordingly. The reliability of the links, which is closely related to the issue of channel dynamics, must be reflected by any sensor network protocol such that erroneous links do not jeopardize the integrity of the key generation process.

Second, due to high node density, scalability is a primary concern. Ad-hoc formation of node clusters [33, 9, 58, 57], hosting collaborative processing, has been a solution in achieving both fault tolerance and scalability. In the cluster formation domain, although the issues of reactivity, energy efficiency, and reliability or fault tolerance have been studied extensively, the security issue has been left unanswered. This has largely hindered the practical deployment of collaborative processing algorithms in many sensor network applications. Consequently, an ad-hoc cluster of nodes is required to establish a joint secret key, and any solid key generation scheme must scale with respect to the number of nodes in a cluster.

The *third* aspect pertains to the scarce energy resources, along with low computation capability, which are always important considerations in security solutions for sensor networks; there is a clear need for conserving energy on each node when adopting a security protocol. In addition to the efficient utilization of energy, its *balanced* consumption across the entire network should be viewed as a primary goal in an aim to prolong the network lifetime.

1.3 Dissertation Outline

This dissertation aims to address the fundamental need for a resilient, scalable, resource-efficient security infrastructure for next-generation sensor networks. In particular, the work focuses on developing, analyzing and implementing public key cryptographic methodologies uniquely designed for WSNs environments. The effort spans contributions on resource-efficient arithmetic derivations of fundamental cryptographic operations. To that end, the different components of this work all contribute to the overall theme of *applied cryptography for resource-constrained environments.*

Chapter 2 provides a literature review with focus on general cryptographic foundations, covering both symmetric as well as public key cryptographic fundamentals. In addition, methods proposed in the context of WSNs in recent years are described. Chapter 3 introduces the self-certified public key generation framework that was developed and implemented. A delay-efficient, self-certified group key generation methodology is presented and analyzed in Chapter 4, while Chapter 5 focuses on the need for denial of service mitigation in resource-constrained sensor networks. In Chapter 6, various light-weight arithmetic algorithms are described and analyzed, constituting a novel framework for performing complex cryptographic functions on low-resource platforms. Finally, in Chapter 7, a summary and concluding remarks are provided.

Chapter 2

Literature Review

2.1 Introduction

Security challenges can be coarsely divided into four closely intertwined categories [62]: secrecy, authentication, nonrepudiation and integrity control. Secrecy, also called confidentiality, means that the information cannot be understood by anyone for whom it was unintended, i.e., keeping information out of the hands of unauthorized users. Authentication deals with determining whom you are talking to prior to revealing sensitive information, i.e., the sender and receiver can confirm each other's identity. Nonrepudiation means that the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information. Integrity makes certain that the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected.

Modern cryptography generally includes two classes of algorithms: those based on symmetric-keys and those based on public-keys. The former uses the same (secret) key for both encryption and decryption; while the latter requires each user to have two (different) keys: a public key, used by the entire world for encrypting message to be sent to that user, and a private key, which the user needs for decrypting messages. The private key can also be used by its owner for signing messages, where the signature can be verified by any party that has the non-secret public key. Public-key cryptographic algorithms are orders of magnitude more complex to implement than symmetric key algorithms. Therefore, in practice all data traffic is encrypted using symmetric-key cryptography (typically, Advanced Encryption Standard or triple Data

Encryption Standard), where public key cryptographic techniques are used just for generating and authenticating the symmetric key used in the symmetric-key encryption/decryption. Unfortunately, due to the extremely scarce resources in sensor networks, public key methods are commonly perceived to be infeasible for sensor nodes.

In the context of wireless sensor networks, much of the work appearing in the literature has focused on symmetric cryptography techniques, as will be later described in more detail. In particular, the notion of key-predistribution plays a key role in existing security solutions for WSNs. To understand these techniques, we next provide an overview of applied cryptographic methodologies.

2.2 Symmetric Cryptography

2.2.1 Introduction

In symmetric cryptography, the same key is used by all parties in both encryption and decryption of the data exchanged (see figure 2-1 for visual illustration). This key is a secret shared only by a designated group of users. The primary advantage of symmetric key cryptography algorithms, such as DES [51] and AES [54], lies in that they are very fast, and hence are used for processing large amounts of data. However, a major disadvantage of these schemes pertains to the fact that the secret key must be agreed upon *prior to the exchange of information* and respective encryption and decryption processes. There are no known ways to generate and agree upon a shared secret key over insecure media in the realm of symmetric key cryptography, hence it is usually combined with Public Key Cryptography (PKC) which provides a solution to this very issue, as will be described in more detail in section 2.3.

2.2.2 The Data Encryption Standard (DES)

Adopted in 1976 by the National Institute of Standards and Technology (NIST) and made publicly available in 1977, the Data Encryption Standard [51] (previously known as the Lucifer algorithm) became a very widely employed technique in a short period of time. As quoted by NIST in 1999 [52], *The goal (of DES) is to completely scramble the data and key so that every bit of the ciphertext depends on every bit of the data and every bit of the key .. (such that)*

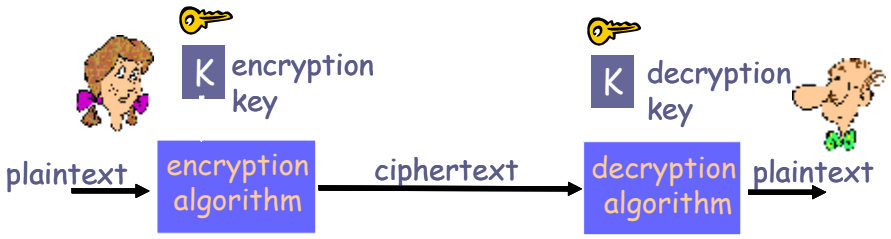

Figure 2-1: The foundations for symmetric cryptography: Alice is encrypting the plain message, and Bob is decrypting the chiphertext, while both use the same shared secret key (redrawn from [42])

there should be no correlation between the ciphertext and either the original data or key.

DES is a symmetric key cryptography method which encrypts and decrypts a 64-bit plaintext with a 64-bit key, although the effective key strength is only 56 bits. The 16 steps (as the number of steps is increased, security of the algorithm increases exponentially) of the DES protocol start with 64 bits of plaintext and ends with 64 bits of ciphertext. As indicated above, the size of the key is 64 bits, but only 56 of them are effective in the encryption process; the least significant bit of each byte is a parity bit and is to be ignored. This bit is set such that all bytes have an odd number of 1's. Next, we briefly review the steps comprising the DES algorithm (see [51] for more details).

Steps for transforming the 64-bit key

The first two phases of the protocol deal with permuting the 64-bit key:

1. The 64 bits are transformed into 56 bits using a permutation called the Permuted Choice 1 ($PC-1$).

2. The new set of 56 bits is transformed into sixteen 48-bit sub-keys called $K_1 - K_{16}$. A subsequent permutation is generated using the Permuted Choice 2 ($PC-2$) scheme. These 16 sub-key are used in the 16 rounds of the DES encryption and decryption process.

Steps for encryption and decrypting data

The following are the steps required to encrypt (and as will be described below, to decrypt) a 64-bit plaintext (see figure 2-2 for details):

1. The plaintext is passed through a permutation phase called the Initial Permutation (IP).

2. Following the Initial Permutation, the 64 bits are substituted (or shuffled) such that the right-most 32 bits are exchanged with the left 32 bits. The right-most 32 bits are identified as R_1 and the left 32 bits are identified as L_1.

3. The permuted and initially substituted text is passed though 16 identical rounds. Each round, i, takes the output of the previous round as its input, and performs the following ($i = 2\text{˜}17$):

   a. R_i and L_i, along with K_i are inputs to a function f.

   b. $L_{i+1} = R_i$

   c. $R_{i+1} = f(L_i, R_i, K_i)$

4. The final substitution consists of exchanging the right-most 32 bits (identified as R_{17}) with the left-most 32 bits (identified as L_{17})

5. The final step is to apply Inverse Initial Permutation (IP^{-1}) to the pre-output. The result is a completely encrypted ciphertext.

In order to decrypt the final ciphertext, the exact same procedure should be applied. The only difference lies in the order of the 48-bit of keys ($K_1 - K_{16}$) that are used. When the identical 16 rounds are applied, the order of the keys is reversed, i.e., instead of using the keys in the order K_1 to K_{16}, they are used in the order of K_{16} to K_1.

Unfortunately, brute force methods have made it possible to significantly reduce the amount of time needed to uncover a DES key. Despite the fact that a 56-bit key is simply not large enough for high security applications, DES is still widely used by many applications. As a consequence of its vulnerabilities, stronger symmetric encryption schemes that rely on similar rationale as DES have been standardized, including the Advanced Encryption Standard (AES) [54] and Triple DES (3DES) [53].

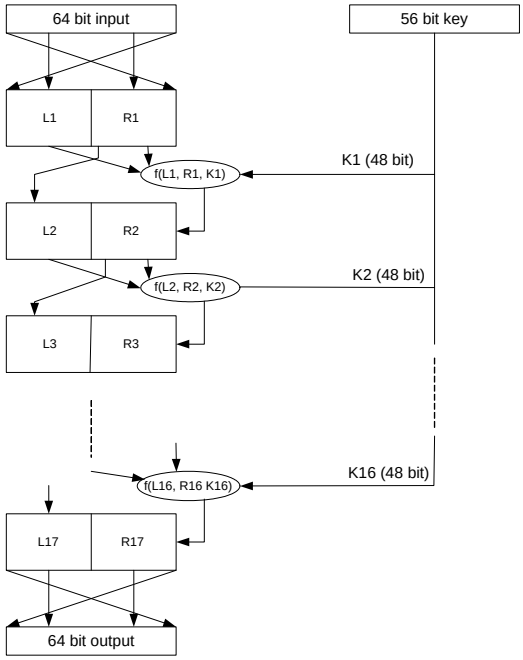

Figure 2-2: The DES algorithm flowchart (redrawn from [42])

2.3 Public Key Cryptography

Public key cryptographic methods were introduced in the late 70's as a novel manner by which data security can be achieved. They inherently rely on asymmetric operations, which complement each other in a mathematically elegant manner. Coarsely speaking, PKC methods are employed by three primary applications:

1. Encryption that is based on a public key (k^+) and decryption based on a private key (k^-) (see figure 2-3 for visual illustration),

2. Signature generation based on a private key, and signature verification based on a public key, and

3. Generation of a symmetric secret key over an insecure channel.

These applications are customarily based on one of two possible intractable mathematical problems: factorizing a large (e.g., 1024-bit) composite integer, or performing a discrete log

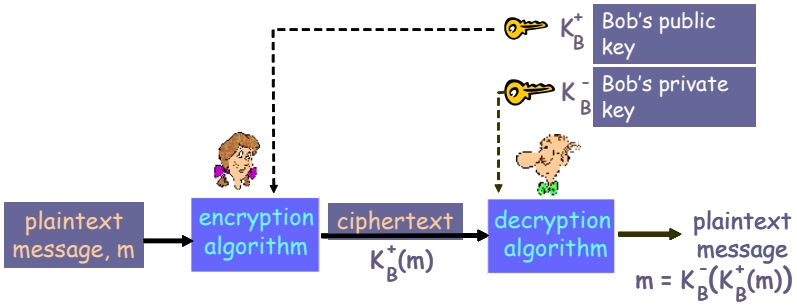

Figure 2-3: The foundations for PKC: Alice is encrypting a plain message, m, using Bob's public key, while Bob is decrypting the chiphertext with his private key (redrawn from [42])

operation. A representative security framework that is based on the factorization complexity is RSA [60]. Applications that are based on the discrete log problem include Elliptic Curve Cryptography (ECC) [41] PKC is approximately three orders of magnitude slower than symmetric cryptography methods for comparable security strength. However, their capacity to operate over insecure media with no prior exchange of information is unmatched, rendering them highly pertinent for a broad range of applications. The subsequent sections provide a brief overview of PKC fundamental building blocks.

2.3.1 The Discrete Logarithm Problem

Let x and n denote two positive real numbers, where N is a finite group and $n \in N$. Let p be a very large prime number. The Discrete Logarithm Problem (DLP) can be stated as follows: Given n, p and $n^x \bmod p$, one is required to compute x. Computing x is extremely difficult (especially for a large enough p), and there are no known efficient algorithms to do so.

2.3.2 The Diffie-Hellman Key Agreement

Diffie-Hellman (DH) key agreement [22] developed in 1976 was a revolutionary protocol which instigated the field of public key cryptography, hence paving the way for *New Directions in Cryptography* (which was, in fact, the title of the original journal publication). Using the DH key exchange method, two users are able to establish a shared secret key while communicating over an insecured channel. The pairwise secret key, as its name suggests, will be known only

to the two parties involved in the key generation process, while preventing any unwanted third party individuals from exposing the key.

Let us assume that the two users, whose purpose is to agree on a shared secret key, are Alice and Bob. Each of the latter holds a different private integer value, x and y, respectively. These private values are considered to be a secret key to each party and can be generated randomly. Both parties must agree on two public parameters: p and a. p is a large prime number and a is an integer smaller than p. Alice takes her secret key, the scalar x, computes the modular exponentiation expression $a^x \bmod p$ and transmits it to Bob (over the unsecured channel). Then, Bob takes his secret key, the scalar y, computes the modular exponentiation expression $a^y \bmod p$, and sends it to Alice. Since the Discrete Log Problem (DLP) applies, none of the users can compute the other party's secret key. The shared key is calculated by each user as the exponentiation $a^{xy} \bmod p$ (Alice calculates the modular exponentiation between the message received from Bob and her private key, i.e., $(a^x)^y \bmod p$ and Bob calculates the exponentiation between the message received from Alice and his private key, i.e., $(a^y)^x \bmod p$.) Due to the commutative attribute of exponentiation, both users end up with the same shared key, $a^{xy} \bmod p$. Here the discrete log problem applies again, i.e., none of the users (Alice or Bob) can compute each others secret key nor can any outside eavesdropper compute the value of the shared secret key $a^{xy} \bmod p$ by knowing the two values $a^y \bmod p$ and $a^x \bmod p$. Figure 2-4 depicts the DH process.

2.3.3 The RSA Methodology

The RSA methodology, which was first published in 1978 by R. L. Rivest, A. Shamir and L. Adleman, is an algorithm for public-key encryption [60]. This algorithm was one of the first used in the world of Public Key Cryptography and is efficient for both encryption and signature applications. The approach of RSA is based on Fermat's Little Theorem [40], and its security is derived from the difficulty of factoring large integers. The keys traditionally used are 1024 bits in size, which offer comparable cryptocomplexity strength to that of an 80-bit symmetric key [5]. Next, we briefly review the RSA algorithm.

The RSA Algorithm

Choosing the Keys

1. Choose two large prime numbers p, q. (e.g., 1024 bits each)
2. Compute $n = p \cdot q$, $z = (p-1)(q-1)$
3. Choose e (such that $e < n$) that has no common factors with z. (i.e., e and z are "relatively prime").
4. Choose d such that $e \cdot d - 1$ is exactly divisible by z. (i.e., $e \cdot d \bmod z = 1$).

The public key (k^+) is the pair (n, e), while the private key (k^-) is the pair (n, d).

Encryption and Decryption Given the public key (n, e) and the private key (n, d) as computed above:

- To encrypt a bit pattern, m (such that $m < n$), compute $c = m^e \bmod n$ (i.e., c is the remainder when m^e is divided by n).
- To decrypt a received bit pattern c, compute $m = c^d \bmod n$ (i.e., m is the remainder when c^d is divided by n).

As can be seen in this case, the claim is that $m = c^d \bmod n \Rightarrow m = (m^e \bmod n)^d \bmod n$. To understand why such an assertion holds, we need to refer to Fermat's Little Theorem [40]: If p, q are prime and $n = p \cdot q$, then: $x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$.

Applying the above to our RSA case, we have $(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$. Using Fermat's Little Theorem we arrive at the deduction that $m^{ed} \bmod n = m^{ed \bmod (p-1)(q-1)} \bmod n$. Since ed is chosen to be divisible by $(p-1)(q-1)$ with remainder 1, we know that $m^{ed \bmod (p-1)(q-1)} \bmod n = m^1 \bmod n$. However, given that the message m is chosen such that $m < n$, we conclude that $m^1 \bmod n = m$, hence $c^d \bmod n = (m^e \bmod n)^d \bmod n = m$.

2.3.4 Elliptic Curve Cryptography (ECC)

In 1985, elliptic curve systems were introduced in cryptography by Neal Koblitz from the University of Washington [40]. Elliptic Curve Cryptography (ECC) is a public key cryptography

framework used for encrypting and decrypting information. Many elliptic curves have been proposed, each with its own cryptocomplexity attributes [17]. Points on an elliptic curve are used in order to create a public key, whereby the number of bits in a key can vary, typically ranging from 79 to 359. ECC is considered to provide the highest security per bit [1]; a 163-bit ECC application provides the same security as a 1024-bit application over a composite integer. This attractive feature of ECC makes it most suitable for sensor network as well as other resource-limited platforms.

An elliptic curve is a finite collection of points in a two-dimensional plane, over $GF(p)$ or over $GF(2^n)$. When operating over $GF(2^n)$ the elements in the field are primitive polynomials. When operating over $GF(p)$, the relation between the (x, y) coordinates of the curve points is specified by the equation $y^2 = \left(x^3 + ax + b\right) \bmod p$, where p is the order of the generating point, i.e., the finite number of points on the curve. Since ECC deals with public key cryptography, there is a need to define the public and private keys. The private key is a scalar and the public key is a point on the curve which is created by multiplying a chosen generating point. As the name suggests, a generating point is a point on the curve that can generate all the other possible points - which constitute a finite group. This point-by-scalar multiplication is the core mathematical foundation behind ECC. As in the case with RSA, the discrete log problem applies here as well, i.e., by knowing the public key (which is a product of the generating point and the private key) and the generating point, it is computationally infeasible to obtain the private key. This property is often referred to as the elliptic curve discrete log problem or ECDLP.

Diffie Helman over an Elliptic Curve

In order to encrypt and decrypt information, the DH method can be employed in order to exchange keys and create a secret key shared by two users. Both parties, Alice and Bob, need to agree on a specific curve and on a point on the curve, $\mathbf{Q}$. Alice holds her secret key, the scalar x, computes the product $\mathbf{Q} \cdot x$ and sends it to Bob. Bob holds his secret key, the scalar y, computes the product $\mathbf{Q} \cdot y$ and sends it to Alice. The shared key is calculated by each user as the product $x \cdot y \cdot \mathbf{Q}$. Alice calculates the product between the message received from Bob and her private key, i.e., $(\mathbf{Q} \cdot y) \cdot x$. Bob calculates the product between the message received from

Alice and his private key, i.e., $(\mathbf{Q}{\cdot}x) \cdot y$. Due to the commutative attribute of multiplications over elliptic curves, both users will carry the same shared key, $x \cdot y{\cdot}\mathbf{Q}$. Since the discrete log problem applies, none of the users (Alice or Bob) can compute each others secret key nor can any outside eavesdropper compute the value of the shared secret key $x \cdot y{\cdot}\mathbf{Q}$ by knowing the two values $\mathbf{Q}{\cdot}y$ and $\mathbf{Q}{\cdot}x$. See figure 2-5 for an illustration of the procedure.

2.3.5 Operations Over Elliptic Curves

A point on the elliptic curve will from here on be denoted by a bold capital letter, e.g. $\mathbf{A}$, whereby the point's coordinates are labeled (x_A, y_A). The following are the basic operations defined over elliptic curves, applicable to the entire field of $GF(p)$.

Definition of a negative point

If the coordinates of $\mathbf{A}$ are $(x_\mathbf{A}, y_\mathbf{A})$, then the coordinates of $-\mathbf{A}$ are $(x_\mathbf{A}, -y_\mathbf{A})$. The point at infinity is denoted by O. This point plays the role of 0 in the sense that $\mathbf{A} + (-\mathbf{A}) = O$, and $\mathbf{A} + O = \mathbf{A}$ (where + denotes a point addition, under the procedures treated next).

Point addition

Let $\mathbf{T}$ and $\mathbf{U}$ be two points on an elliptic curve. The coordinates $(x_\mathbf{R}, y_\mathbf{R})$ of $\mathbf{R} = \mathbf{T} + \mathbf{U}$ (for $x_\mathbf{T} \neq x_\mathbf{U}$) are calculated as follows:

$$b = (y_\mathbf{T} - y_\mathbf{U}) \cdot (x_\mathbf{T} - x_\mathbf{U})^{-1} \bmod p$$

$$x_\mathbf{R} = \left(b^2 - x_\mathbf{T} - x_\mathbf{U}\right) \bmod p$$

$$y_\mathbf{R} = [(x_\mathbf{U} - x_\mathbf{R}) \cdot b - y_\mathbf{U}] \bmod p$$

Point doubling

Let $\mathbf{S}$ and $\mathbf{U}$ be two points on an elliptic curve. The coordinates $(x_\mathbf{S}, y_\mathbf{S})$ of $\mathbf{S} = \mathbf{2} \cdot \mathbf{U}$ are calculated as follows:

$$c = (3 \cdot (x_\mathbf{U})^2 + a).(2 \cdot y_\mathbf{U})^{-1} \bmod p, \text{ (a is the scalar defined on the curve)}$$

$$x_\mathbf{S} = c^{-1} \bmod p$$

$$y_\mathbf{S} = [(x_\mathbf{U} - x_\mathbf{S}) \cdot c - y_\mathbf{U}] \bmod p$$

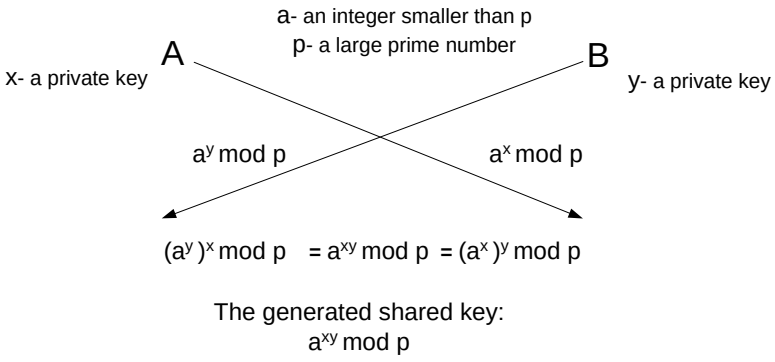

Figure 2-4: The Diffie-Hellman key agreement process

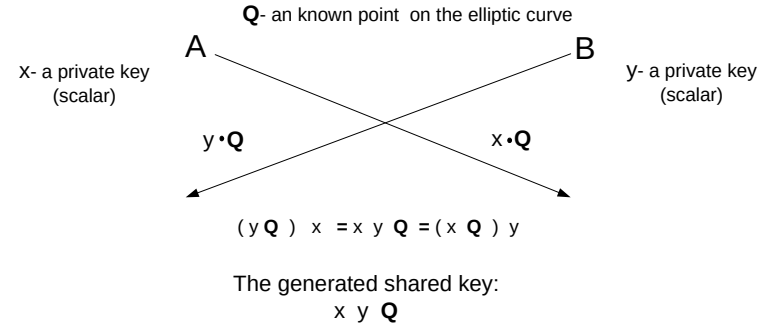

Figure 2-5: Generating a shared secret key using the Diffie Helman method over an elliptic curve

Multiplying a curve point by a scalar

As indicated previously, a multiplication between a point on a curve and a scalar is the fundamental calculation in the world of ECC, hence the importance of understanding it. Let k be an n-bit scalar. k_i denote the i-th bit in the binary representation of k starting with the least significant bit, where $i = 0, 1, \ldots, n-1$. Multiplying a point on a curve by a scalar yields a new point on the curve. Assuming that $\mathbf{L}$ is a point on a curve, the curve point $\mathbf{C} = k\mathbf{L}$ can be calculated by the following double-and-add algorithm. It should be noted that doubling a point and point addition are executed as described above.

$\mathbf{C} = 0$
for $i = 0$ to $n-1$
    $\mathbf{L} = 2\mathbf{L}$
    if $k_i = 1$ then
        $\mathbf{C} = \mathbf{C} + \mathbf{L}$
    end
end

This process is a basic shift-and-add procedure, where shifted (doubled) values of the curve point $\mathbf{L}$ are generated and added to an accumulator based on the binary representation of the multiplier k. An alternative version of a double-and-add process is to scan the bits of k starting with the most significant bit. In this case, the dynamic value of $\mathbf{C}$ is doubled, rather than the fixed point $\mathbf{L}$.

2.3.6 Example of an Elliptic Curve Over $GF(p)$

We conclude this section by providing an illustration of an elliptic curve. As indicated above, the general structure of an elliptic curve over $GF(p)$ is: $y^2 = \left(x^3 + ax + b\right) \bmod p$. Let us consider an example in which $a = 1$, $b = 0$ and $p = 23$, i.e., we are referring to the curve: $y^2 = \left(x^3 + x\right) \bmod 23$. There are 23 points satisfying this equation: $(0, 0)$, $(1, 5)$, $(1, 18)$, $(9, 5)$, $(9, 18)$, $(11, 10)$, $(11, 13)$, $(13, 5)$, $(13, 18)$, $(15, 3)$, $(15, 20)$, $(16, 8)$, $(16, 15)$, $(17, 10)$, $(17, 13)$, $(18, 10)$, $(18, 13)$, $(19, 1)$, $(19, 22)$, $(20, 4)$, $(20, 19)$, $(21, 6)$ and $(21, 17)$. Figure 2-6 depicts the actual curve derived.

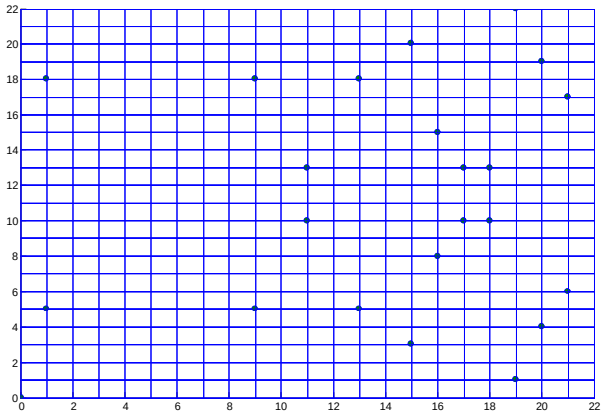

Figure 2-6: Illustration of an elliptic curve over $GF(p)$, where $p = 23$ and the curve is $y^2 = \left(x^3 + x\right) \bmod 23$

2.4 Key Pre-Distribution Schemes in WSNs

Numerous schemes have been proposed in the literature for pairwise and group key agreement. However, as shown by Carman *et al.* [20], most are not suitable for sensor network environments due to lack of information regarding the network topology prior to deployment and the resource limitations of the sensor nodes. Therefore, a mainstream thrust for secret sharing among n sensor nodes has been *key pre-distribution* schemes. Many symmetric key pre-distribution methods have been studied in recent years, with emphasis on multicast and broadcast communication [15], [31], [16]. It is noted that traditional key pre-distribution schemes, such as sharing a single key for the entire network or assigning a unique key to each pair of nodes, are not viable solutions in sensor networks. The single key approach would result in compromising all communication links in the network if a single node is captured by the adversary. The pairwise key sharing for every two nodes has the advantage that capturing any node by the adversary does not directly affect the security of any link between two non-compromised nodes. However, the pairwise key approach requires the storage of $n - 1$ keys in each node (which is unduly large given the memory limitation of sensors). Moreover, it would be hard to scale the network (revocation or appending of sensor nodes) [20].

To address these drawback, Eschenauer and Gligor [28] proposed a random key pre-distribution

scheme: before deployment each node is loaded with a subset of a large key pool. A *shared-key discovery* phase takes place during the initialization in the operation environment where every node discovers its neighbors within the wireless communication range with which it shares a key. This can be simply achieved by storing the identifiers of keys in each sensor node (prior to deployment) and broadcasting these identifiers to the adjacent nodes during the shared-key discovery phase. A link exists between two nodes only if they share a key (from their stored key ring) and all communication on that link is secured by link encryption. Trade-offs can be made between sensor-memory, cost and connectivity, and design parameters can be adapted to fit the operational requirements of a particular environment. It has been shown that this scheme is superior to traditional key pre-distribution schemes. Based on the basic scheme in [28], Chan *et al.* [19] proposed a q-composite random key pre-distribution scheme, which increases the security of key setup such that an attacker must compromise many more nodes to achieve a high probability of compromising communication links. The difference between the q-composite scheme and the scheme in [28] is that at least q common keys ($q \geq 0$) are needed to establish secure communications between a pair of nodes. It is shown that by increasing the value of q, network resilience against sensor-node capture is improved. However, as the number of compromised nodes increases, the fraction of affected pairwise keys increases rapidly. As a result, a small number of compromised nodes may affect a large fraction of the communication links. Du et al. [23] combined the random key pre-distribution method with the classical method of Blom's key pre-distribution. The goal of their scheme is to increase network's resilience against node capture without using more memory. The drawback of this scheme is the computation overhead that occurs at each node and the oversimplified random graph model. Similarly, Liu and Ning [21] developed a pairwise key establishment using the polynomial-based key pre-distribution protocol and probabilistic key distribution. It has been indicated that the scheme has several advantages. In particular, unless the number of compromised sensors that share a common polynomial exceeds a threshold, the sensor-node capture does not lead to the compromise of the links established by non-compromised nodes. Peer intermediaries for key establishment (PIKE) have been proposed in [18] similar to the 2D grid-based scheme in [21]. However, the main disadvantage of this method is the size of the key ring stored in the memory of each sensor. Every sensor has to store $2(m-1) = O(\sqrt{n})$ keys while in [21], the memory

requirement is $O(t)$ where t, the degree of the polynomials, is independent of the network size. Moreover, we need to assume that none of the involving nodes are compromised during the key-path discovery.

Key pre-distribution means the following: each sensor node before deployment is pre-configured with a subset of keys (called a key ring) along with their associated identifiers. After the (random) deployment of sensor nodes, a shared-key discovery phase takes place by which two nodes within the communication range of each other find out whether they share at least one common key [28]. Then, each node uses their shared key (we refer as "direct link key") to establish a secure communication link. We assume that the sensor nodes are not mobile and they have similar computation and communication capabilities. The nodes are also limited in memory and power. We call a network "connected" if there exists a communication path consisting of hops between every two nodes. The robustness of sensor networks is linked to their connectivity. Without such connectivity, the network loses proper functionality. The communication range of the sensors can be varied in order to achieve connectivity. However, the communication range should also be kept minimal because of very limited power supply. In the attack model, we assume that if a node is compromised, all the information (including the stored keys in the key ring of the node) within the node will also be compromised.

Despite their popularity, key pre-distribution schemes offer partial solution with respect to scalability, cryptographic robustness and the ability to append and revoke security attributes. For example, scalability is limited, since the probability of two or more nodes sharing a pre-distributed key decreases rapidly as the number of nodes increases. This results in a need for a key discovery process, in which nodes communicate with other nodes in order to identify a joint secret key – a process that necessitates additional resources.

The cryptographic robustness is also lacking in pre-distribution schemes, as reflected by two aspects: first, the use of static key rings which are assigned to the nodes do not facilitate dynamic key generation, i.e. the generation of a new secret key per session, thereby reducing the cryptographic strength offered; second, by capturing a node an adversary party may be able to decrypt data exchanges between other nodes in the network (given that the nodes utilize keys that are present in the captured node).

2.5 Self-Certified Key Establishment

A self-certified key establishment is a key distribution technique in which the authentication process is embedded inside the key generation process. Since there is no need for an authentication process preceding the key generation process (as is usually the case), this procedure is very efficient, and hence can be practical for WSN applications.

The basic techniques introduced in this section are derived from the work in [7]. The mathematical foundations rely on ECC cryptographic techniques pertaining to operations over a finite group of points in which the discrete log problem applies. Diffie-Hellman (DH) key exchange is a cryptographic protocol, which allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. Denote a "point" on an elliptic curve by a capital letter in bold font (e.g., **P)**. Multiplication of a point by a scalar (e.g., $s \times \mathbf{P}$) is commonly referred to as an exponentiation, in which s is called the exponent.

ECC operations are based on the existence of a generating point $\mathbf{G}$, with an order $ord\mathbf{G}$, which is known to all parties. The private and public keys are issued by the CA to all users in the network. The CA holds a pair of keys, a private key which is a scalar denoted by d, and a public key, which is a point denoted by $\mathbf{R}$, where $\mathbf{R} = d \times \mathbf{G}$. Let ID_i denote the ID or any other relevant attributes of a user N_i. The notation $H(s, \mathbf{P})$ refers to a scalar obtained by hashing a scalar s and a point $\mathbf{P}$. In general, the following framework applies to all key generation methodologies presented here. First, key-issuing takes place, whereby a user acquires off-line a set of public and private keys. Next, a joint key is established online via self-certified DH, followed by key confirmation.

2.5.1 Key-issuing Procedures

The CA issues to user N_i a private key (x_i), and the public key ($\mathbf{U}_i$). The key-issuing procedure is thus performed as follows:

1. The CA generates a random scalar h_i

2. The CA then generates user N_i's public and private keys as follows:

$$\mathbf{U}_i = h_i \times \mathbf{G},\ x_i = [H(ID_i, \mathbf{U}_i) \times h_i + d] \mod ord\mathbf{G} \tag{2.1}$$

3. The CA issues the values x_i and $\mathbf{U}_i$ to N_i;

4. N_i can establish the validity of the values issued to it by checking whether

$$x_i \times \mathbf{G} = H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}$$

2.5.2 Self-certified Fixed Key-establishment

A self-certified DH fixed-key-establishment is achieved by the following two steps: (1) N_i and N_j exchange the pairs $(ID_i, \mathbf{U}_i)$ and $(ID_j, \mathbf{U}_j)$, respectively, and (2) N_i and N_j generate the session-key,

$$\begin{aligned} K_{ij} \text{ (generated by } N_i) &= x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] \\ K_{ji} \text{ (generated by } N_j) &= x_j \times [H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}]. \end{aligned} \tag{2.2}$$

See figure 2-7 for details

Key confirmation should now follow, where N_i and N_j should encrypt and decrypt a test message, using their keys K_{ij} and K_{ji} , and verify that they actually share the same key. The two keys are expected to be identical, having the value $x_i \times x_j \times G$. Verifying, by an independent key-confirmation procedure, that the keys generated by the two users are indeed equal establishes their correct identities. This closes the trust loop controlled by the CA. Key confirmation can be executed by using, for example, DES.

2.5.3 Self-certified Ephemeral Key-establishment

A self-certified DH ephemeral key-establishment is achieved by the following steps: (1) N_i and N_j generate uniform i.i.d. random pv_i and pv_j, respectively, (2) N_i calculates the ephemeral value $\mathbf{EV}_i = pv_i \times \mathbf{G}$, while N_j calculates the ephemeral value $\mathbf{EV}_j = pv_j \times \mathbf{G}$, (3) N_i and N_j exchange the values $(ID_i, \mathbf{U}_i, \mathbf{EV}_i)$ and $(ID_j, \mathbf{U}_j, \mathbf{EV}_j)$, respectively, and (4) N_i and N_j

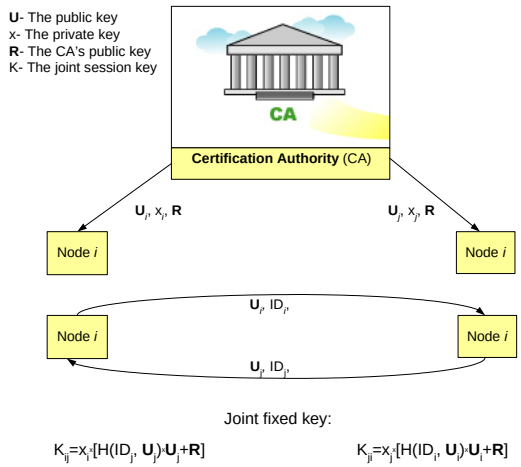

Figure 2-7: Self-certified ECC-based fixed key generation process

generate the ephemeral session key,

$$K_{ij} \text{ (generated by } N_i) = pv_i \times [H(ID_j, \mathbf{U}_j)] \times \mathbf{U}_j + \mathbf{R}] + (x_i + pv_i) \times \mathbf{EV}_j$$
$$K_{ji} \text{ (generated by } N_j) = pv_j \times [H(ID_i, \mathbf{U}_i)] \times \mathbf{U}_i + \mathbf{R}] + (x_j + pv_j) \times \mathbf{EV}_i \qquad (2.3)$$

As specified for the fixed-key case, key confirmation (which can be executed by using DES for example) should now follow. That is, N_i and N_j should encrypt and decrypt a test message, using their keys K_{ij} and K_{ji}, and verify that they actually share the same key. The two keys are expected to be identical, having the value $pv_i \times x_j \times \mathbf{G} + x_i \times pv_j \times \mathbf{G} + pv_i \times pv_j \times \mathbf{G}$. Verifying that the keys generated by the two users are indeed equal, establishes their correct identities.

It is very important to emphasize that in both the ephemeral and fixed-key generation procedures, the authentication process is indeed imbedded in the key generation process. This realization pertains to the fact that K_{ij} and K_{ji} will indeed be identical only if the proper private keys, x_i and x_j, are used respectively. If users N_i and N_j hold the correct secret key (issued only to them by the CA) the key confirmation process will indeed confirm the proper identification as well as the fact that their keys are indeed identical.

Chapter 3

Self-Certified Public Key Generation for Resource-Constrained Sensor Networks

In this chapter we describe a comprehensive ECC-based self-certified key establishment methodology, suitable for WSN environments. Furthermore, a method for generating a joint secret key between an ad-hoc cluster of nodes is introduced. Although group key establishment based on public key cryptography has been considered in the literature [39], there is little to no treatment of the issue of authentication. In fact, a common assumption made by these schemes is that an authentication mechanism is already available, while the proposed method also concerns the efficient integration of self-certified authentications.

In an effort to effectively distribute the computational load between the nodes, we propose to partition the self-certified key-establishment process into secure and non-secure operations. This enables off-loading the non-secure operations from a node participating in the key-establishment process to available neighboring nodes which do not. Such distribution of the computational effort yields improved load-balancing, shorter execution times and more homogeneous power consumption across the network.

3.1 Adopting a Load-balanced Key-establishment Procedure

The self-certified fixed and ephemeral key establishments described in sections 2.5.2 and 2.5.3, respectively, can be used as basis for key generations in WSNs. As shown in section 2.5.2, a primary attribute offered by the method of self-certified fixed-key establishment lies in the number of exponentiations required to calculate the value $x_i \times x_j \times G$. As indicated, node N_i calculates $K_{ij} = x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] = [x_i \times H(ID_j, \mathbf{U}_j) \bmod ord\mathbf{G}] \times \mathbf{U}_j + x_i \times \mathbf{R}$. The value $x_i \times \mathbf{R}$, which utilizes fixed values (where x_i denotes node's N_i private key and $\mathbf{R}$ denotes the CA's public key), can be pre-calculated and stored by N_i. Therefore N_i is able to calculate its session-key by the single online exponentiation $[x_i \times H(ID_j, \mathbf{U}_j) \bmod ord\mathbf{G}] \times \mathbf{U}_j$, generating a certified key using only one exponentiation. In comparison, a standard fixed-key establishment requires three online ECC exponentiations (two for validating a certificate and one for the key generation). Similar rationale follows for the case of ephemeral key generation, as described in 2.5.3, Note that the calculations performed by N_i and N_j are

$$\begin{aligned} K_{ij} &= [pv_i \times H(ID_j, \mathbf{U}_j) \bmod ord\mathbf{G}] \times \mathbf{U}_j + (x_i + pv_i) \times (\mathbf{EV}_j + \mathbf{R}) - x_i \times \mathbf{R} \\ K_{ji} &= [pv_j \times H(ID_i, \mathbf{U}_i) \bmod ord\mathbf{G}] \times \mathbf{U}_i + (x_j + pv_j) \times (\mathbf{EV}_i + \mathbf{R}) - x_j \times \mathbf{R} \end{aligned} \tag{3.1}$$

The pre-calculation and storage of $x_i \times \mathbf{R}$ (which is fixed for all key-establishment procedures in which N_i participates) would enable N_i to calculate its session-key by performing only two online exponentiations $[pv_i \times H(ID_j, \mathbf{U}_j) \bmod ord\mathbf{G}] \times \mathbf{U}_j$ and $(x_i + pv_i) \times (\mathbf{EV}_j + \mathbf{R})$. This is preceded by the off-line calculation of $\mathbf{EV}_i = pv_i \times \mathbf{G}$, executed at each session using a different pv_i. The latter can be carried out at any stage prior to establishing a communication session with N_j.

3.1.1 Off-loading of Computational Efforts to Neighboring Nodes

Off-loading non-secure tasks from a component having limited resources to an assisting node is not new. This approach is used for example in RSA key generation [7] and in broadcast encryption [10]. In this work, the new approach to load-balancing among WSNs motes is based on manipulations with Diffie-Hellman key-establishment mathematics.

Both fixed-key and ephemeral-key establishments are treated here. Their suggested employ-

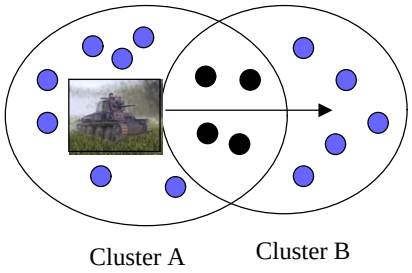

Figure 3-1: Illustration of two clusters established in accordance with a moving target

ment in WSNs is illustrated in figure 3-1.

In the interest of distributing the power consumption across the sensor network, we employ an off-loading technique in which nodes assist other nodes by performing part of the required calculations. In the context of security operations, we must prove that calculations that are off-loaded, and are subsequently transmitted over potentially eavesdrop-prone channels, do not jeopardize the trustworthiness of the process.

As shown in section 2.5.3, the ad-hoc operations executed during the ephemeral key-establishment are $[pv_i \times H(ID_j, \mathbf{U}_j) \bmod ord\mathbf{G}]$ and $(x_i + pv_i) \times (\mathbf{EV}_j + \mathbf{R})$. The first must be executed at the node N_i as it contains the private ephemeral value pv_i. Assisting neighboring nodes (not included in the ad hoc cluster, but with proximity to it) will calculate the value $(x_i + pv_i) \times (\mathbf{EV}_j + \mathbf{R})$. It should be noted that all nodes are assumed to have knowledge of $\mathbf{R}$. While x_i and pv_i are secret, their sum does not disclose their values. Moreover, even though x_i is fixed, pv_i never repeats itself. In other words, the secret key x_i is masked with the random noise pv_i. It is further noted that the neighboring assisting node is not necessarily trusted in delivering a correct answer. The assisting node merely performs mathematical processing and does not issue any decisions. An attempt to send a misleading result by an assisting node will be detected during the key confirmation step.

3.1.2 Communication Framework

The approach taken in this section is that of exploiting spatial off-loading of calculation tasks needed to establish a joint key between two nodes. Available nodes, not included in the current

cluster, assist other nodes by concurrently performing portions of the necessary computations. Figure 3-2 illustrates a basic network topology in which nodes A and B wish to establish a joint ephemeral key, with the assistance of nodes C and D. The following protocol outlines the process by which a joint key is established between the two nodes:

1. A broadcasts a message to B, which includes a unique ID number, ID_A, requesting to establish a joint key.

2. B replies with a confirmation broadcast message containing ID_B.

3. A and B exchange $(ID_A, \mathbf{U}_A, \mathbf{EV}_A)$ and $(ID_B, \mathbf{U}_B, \mathbf{EV}_B)$, respectively.

4. Nodes A and B look for assistance from other neighboring nodes not included in their cluster. In this case, they will seek assistance from nodes C and D, by sending them, respectively, assist request (AST_REQ) messages containing ID_C and ID_D. Neighboring nodes C and D, receive the assist request messages from nodes A and B, respectively, and reply, if possible, by sending assist acknowledgement (AST_ACK) messages indicating their availability to take part in the calculation process. As part of the AST_ACK message, both C and D include unique ID numbers, ID_C and ID_D.

5. Upon receiving AST_ACK messages from C and D, A and B respond by sending data that includes ID_{CA}, $(x_A + pv_A)$,.$(\mathbf{EV}_A + \mathbf{R})$ and $ID_{DB}, (x_B + pv_B)$, $(\mathbf{EV}_B + \mathbf{R})$ respectively.

6. C and D send the result of their computation processes, i.e.,$(x_i + pv_i) \times (\mathbf{EV}_j + \mathbf{R})$ and $(x_j + pv_j) \times (\mathbf{EV}_i + \mathbf{R})$ respectively, to A and B, respectively.

7. The joint key is established, followed by key confirmation.

The described off-loading concept suggests that assisting nodes are self organized in the sense that there is no centralized entity pairing nodes. Therefore, a key question is how are assisting nodes identified. The proposed method relies on the use of a single *weight* value, w_i, calculated at each node i, reflecting on its availability to assist. For example, the weight can be proportional to the remaining energy of the node. The larger the weight the higher the node's availability to assist.

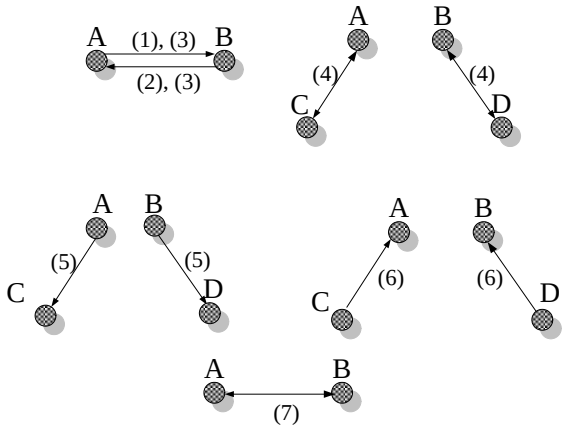

Figure 3-2: Network protocol employed by the proposed key establishment methodology. Nodes A and B, who aspire to establish a joint key, are assisted in calculations by neighboring nodes C and D

Let us assume node N_i receives a request for assistance message, AST_REQ, from a neighboring node, and its current weight is w_i. Node N_i then waits for $\exp(-\frac{w_i^2}{2\sigma^2})$ amount of time to respond with an acknowledgement message, AST_ACK. The parameter (σ) controls how fast the Gaussian function decreases with respect to the weight (w_i). If, during the waiting period, a different neighboring node, say N_j, broadcasts an AST_ACK message, N_i will discard the AST_REQ and N_j is assumed to serve as the assisting node. However, if N_i does not receive any AST_ACK indicating assistance before the waiting time expires, N_i will become the assisting node.

If the node that requests assistance does not receive any AST_ACK within a certain amount of time, t_{wait}, then this node will perform all calculations. Such scenario might occur when all neighboring nodes are at a low-energy level or the communication link has errors.

3.2 Group-key Establishment based on Pairwise DH Key Establishment

3.2.1 Formation of a Group Key

It is next shown how a group of m nodes generates a secret session key K_s joint to all nodes in the group and not attainable to any party outside the group. In this respect it is noted that the self authentication of the DH keys is based on the identity, ID_s, of the participants. These identity values can also be associated with attributes of nodes, rather than their explicit identities. For example, they can be associated with parameters that specify the meaning of the group. That is, nodes that do not posses appropriate parameters allowing them to participate in the group cannot force themselves into the group.

Let the nodes in the group be indexed in a chain, where node N_i generates two DH keys, one jointly generated with node N_{i-1} and one with $N_{i+1}, i = 0, 1, \ldots, m-1$. Although this is not a necessity, the indexing is cyclic. That is, N_{m-1} and N_0 also generate a joint key. For simplicity, let us further assume that m is even. These $2m$ DH keys can all be generated within two time slots. Let K_{i+} denote the DH key joint to nodes N_i and N_{i+1}, generated during the first time slot for even i's, and K_{i-} denote the DH keys generated during the second time slot for odd i's. This way, during each slot, every node is busy generating a joint DH key with exactly one other node.

Based on the fact that each node possesses two DH keys, one joint to the preceding node in the chain and one joint to the subsequent node (where N_{m-1} and N_0 are considered to be consecutive), the secret session key K_s, joint to all members in the group, is then generated as follows. A certain node N_j in the group (N_j can be an arbitrary node, or a node with some distinct preferences such as the cluster head or group leader) generates a random K_s. It encrypts K_s with K_{j+} and sends the ciphertext to N_{j+1}. Node N_{j+1} decrypts the ciphertext, as it also has K_{j+}, thereby recovering K_s. It then encrypts K_s with the DH key joint to N_{j+1} and N_{j+2}, etc. This way, K_s securely propagates in the chain, by decryption and encryption operations taking place at each node. K_s finally gets back to the originator N_j, who verifies that the received K_s is identical to the original.

Although calculations are carried out concurrently by the odd and even nodes, we must

consider the fact that transmission of information may be done sequentially, since the same wireless channel is shared by all nodes. Letting t_{access} and t_x denote the expected channel access time and transmission/reception times, respectively, the aggregate time consumed by the group key establishment process, T_{gk}, can be expressed as

$$T_{gk} = 2m(t_{access} + t_x) + t_{DH}, \tag{3.2}$$

where t_{DH} is the overall time required to perform the actual DH calculations. One should note that the access and transmission times are expected to be in the order of milliseconds, while the DH related computations are in the order of seconds (shown for MICA2 sensor platforms [45] and Intel 2 sensor platforms [6]). To that end, the fact that communications are done sequentially has little impact on the overall delay of the group key establishment process.

It should be noted that the encryption/decryption functions performed at each node (when protecting the joint key K_s) consist of symmetric operations which can be based on standard procedures like DES or AES. However, let us also consider the case where this operation is a simple exclusive-OR (XOR) operation between K_s and K_{j+}. That is,

$$c_j = K_s \text{ XOR } K_{j+}, \tag{3.3}$$

where c_j is the ciphertext sent from N_j to N_{j+1}. Node N_{j+1} then performs the following to propagate K_s to N_{j+2} (note that N_j and N_{j+1} share the same key K_{j+}, and N_{j+1} and N_{j+2} share K_{j+1-}),

$$K_s = c_j \text{ XOR } K_{j+} \text{ XOR } K_{j+1-}. \tag{3.4}$$

As the nodes finally obtain K_s, it is noted that all pairwise DH keys can also be known to the nodes in the group by simply applying XOR to K_s and all ciphertexts. A related question, which raises a strategic consideration, is what kind of a threat can be posed by this procedure. After all, if the members of the group finally know the joint secret key, K_s, they might as well know the individual DH keys. This surely holds if the DH keys expire at the end of the session that utilizes the key K_s.

A very important observation concerns the issue of propagating the group key via XOR

operations, as mentioned above. It is imperative that key confirmation (as shown in section 2.5.2 and 2.5.3) will precede the actual propagation of the key. In other words, N_j will propagate K_s by sending K_s XOR K_{i+} only if the key confirmation between N_j and N_{j+1} was successful. If not, N_j must overcome the obstacle and create an immediate joint key with N_{j+2}, thereby enabling the continuation of the chain (here we assume that N_1, the first node generating K_s, is the cluster head and is not malicious). In these cases, whereby key confirmation is not successful, (suggesting that N_{j+1} might be malicious), the group key (K_s) will be propagated along with the ID of the potentially malicious node. By the end of the distribution process, the IDs of all potentially malicious nodes in the cluster will be known to the legitimate nodes (holding the desired group key).

This ring-based topology group key establishment (presented in this section) can be significantly improved, ensuring higher fault tolerance. Various methods using other topologies, such as a tree topology, have been published, [25], [38]. Chapter 4 in this dissertation describes an improvement of this group key generation in terms of delay time, independent of the pairwise key establishment described in section 3.1 and without the need of a specific topology.

3.2.2 Countering Possible Attacks

Several possible attacks, including Denial of Service (DoS) should be addressed. Two forms of DoS attacks can occur impacting the effectiveness of the off-loading framework. In the first type of DoS attacks, one or more malicious nodes, pretending to be nodes seeking assistance, can continuously send requests to neighbors thereby draining their energy. In these cases, the weight value generated at each node when receiving the request message can alleviate the DoS attack, as the more off-load calculation requests a node receives, the more energy it will waste, and consequently, the longer the waiting time will be. This results in lower possibility of being served as the assistant node. A possible problem lays in the option that a malicious node can alway try to offer help first, and then will not. In this case the node will have to do all of the calculations by himself.

The other type of DoS attacks can happen when the malicious nodes pretend to be the assisting nodes by always generating the highest weight value and thus always responding the quickest to request messages, and then returning incorrect calculation results. In these cases,

however, the key confirmation process is inherently able to counter it. It is noted that following the establishment of a shared key, key confirmation follows. That is, nodes N_i and N_j encrypt and decrypt a test message to verify that they have the same key. If indeed the keys are identical, both nodes can be trusted (i.e. each node can trust its counterpart). If $K_{ij} \neq K_{ji}$ then it is clear that either one (or both) of the assisting nodes was malicious, or that there was an innocent error (for example due to link error). In both cases, if the key is not confirmed, then the joint key needs to be re-established. Under the scenario where a group key needs to be created, the two nodes that did not generate a joint key will be eliminated from the group.

3.3 Cryptocomplexity Analysis and Experimental Results

3.3.1 Cryptocomplexity Analysis

The unique nature of WSNs merits a brief discussion on cryptocomplexity. Let the term MIPS denote million of instructions per second. We rely on the fact that one MIPS computer performs about 2^{40} elliptic curve additions per year, which translates to approximately 80 iterations per second [41]. For a key of n bits, a rough estimation of the number of additions needed for solving an elliptic curve discrete logarithm problem (ECDLP), is $2^{n/2}$. Relying on the latter, we present a cryptocomplexity summary of the key sizes discussed. Here we use 160-bit keys (over a field size of 163 bits).

Recent challenges for solving the ECDLP over a field size of 109 have been issued [1]. In April 2004, the challenge was met and the ECDLP key was solved. The effort involved four months and 9,500 CPUs. In light of the fact that the time frame for the validity and confidentiality of WSNs data is typically in the order of at most days, ECC-based key generation offers a high level of security.

3.3.2 Energy Consumption and Pairwise Key-establishment Time

Implementations of the Key-establishment and measurements of energy consumption and time have been measured on two different platforms, the TelosB [3] and the Intel Mote 2 [6], both described below.

Figure 3-3: Crossbow/UC-Berkeley's TelosB sensor platform

Implementations on the Crossbow, Inc. TelosB Platform

We have implemented the presented key establishment procedures on a TPR 2400CA TelosB network sensor module [3], developed by the University of California, Berkeley, and manufactured by Crossbow, Inc. The TelosB platform's microcontroller unit is a 16-bit MSP430, an ultra low power controller manufactured by Texas Instruments, running at 4MHz and consuming 1.8mA. The transmission rate is 250 kbps, consuming 19.5mA when the radio is active. The unit draws a voltage of 3V. Two such motes were used in the experimental setup. One originated the request for key exchange, transmitting the necessary information to the other, while the other responded with the required calculations and transmitted back the data. The same transmissions and calculations were symmetric with the second node as an initiator. For the purpose of calculating the energy consumptions, all metrics were measured both for transmitting and receiving data as well as for the computational component.

Given the variance that exists with regards to the communication needs associated with each scheme, a brief discussion on the implications of the communication load is in order. From a protocol stack perspective, the security layer can be viewed as part of the application layer. The topic of media access control (MAC) protocol in WSNs has received much attention in recent years, primarily due to its unique characteristics. When comparing results to other schemes, this work views the issue of efficiently accessing the media (i.e. balancing sleep and active transmission/reception periods) as the responsibility of the MAC layer. For that reason,

Table 3.1: Time and energy consumptions for scalar-point multiplication and radio transmission on the TelosB sensor platform

Scalar Point Multiplication			
EccM			
Time (seconds)	Voltage (v)	Current (mA)	Energy (mJ)
32.5	3	1.8	184
TinyEcc			
Time (seconds)	Voltage (v)	Current (mA)	Energy (mJ)
16	3	1.8	76
Radio Transmission (including a 7 byte header)			
Time (msec)		Energy (mJ)	
~15		0.038	

the analysis is driven by measurements reflecting the consumption of the security layer and do not consider inefficiencies (i.e. periods of unnecessary "active" periods) introduced by the MAC layer. This is a valid perspective given that all security protocols rely on efficient MAC layer functionality.

The ECC key sizes used in our measurements is 160 bits. Its cryptographic complexity is, equivalent to that of 1024-bit RSA. As discussed previously, both these values are specified by the National Institute of Standards and Technology (NIST) Computer Security Resource Center [26]. EccM, the original code provided by Malan *et al.* [45] , which was designed for the 8-bit MICA2 mote [2], was revised and optimized for TelosB implementation (see figure 3-3). Modifications to the code were carried out in order to exploit the 16-bit based operations supported by the MSP430. The revised code yielded execution of an ECC scalar-point multiplication in 32.5 seconds for 160-bit keys. Memory needs were about 20Kbytes of ROM and 1500 bytes of RAM. These self-certified algorithms (both fixed and ephemeral) were also implemented using functions taken from the TinyECC package [49]. In this case the results were even more encouraging. The code yielded execution of an ECC scalar-point multiplication in only 14 seconds for 160-bit keys.

Table 3.2 describes the time and energy consumption for scalar-point multiplication and radio transmission on the Crossbow/UC-Berkeley's TelosB sensor platform (using a field size of 163 bits). Note that the actual time the radio is in use is greater than 15 msec (since it takes time to also power up and power down). Calculations have been done using both the EccM code and the TinyEcc code.

Table 3.2: The time computed for establishing an online pairwise fixed and ephemeral key on the TelosB sensor platform

Key type	*Number of online exponentiations*	*Calculation time (seconds)*
Fixed key- EccM	1-calculated by a node in the cluster	34
	0-offloaded	0
Fixed key- TinyEcc	1-calculated by a node in the cluster	18
	0-offloaded	0
Ephemeral key- EccM	1-calculated by a node in the cluster	34
	1-offloaded	34
Ephemeral key- TinyEcc	1-calculated by a node in the cluster	18
	1-offloaded	18

Clearly, for the same key size, the energy consumed by radio transmission is **three orders of magnitude** lower than the energy consumed by calculating a scalar-point multiplication. That is, the transmission overhead is negligible compared to the computational efforts, strongly advocating the off-loading approach pursued in this section.

As indicated above, a node in a cluster needs to execute one exponentiation in order to perform both online fixed and ephemeral key-establishments (whereby in the latter, a second online calculation is off-loaded to a neighboring node).

In the model considered, a node is either part of the cluster or an assisting node. Hence, assisting nodes do not take part in active information gathering and collaborative data processing. Moreover, a node will only assist a single other node at any given time. To that end, the overall gain achieved in the cluster by performing off-loading for a single pair of nodes is linear with respect to the number of key pairs established.

Table 3.2 describes the time for establishing an online pairwise fixed and ephemeral key on the Crossbow/UC-Berkeley's TelosB sensor platform (using a field size of 163 bits). See figure 3-4 for self-certified time requirements for TelosB using the TinyEcc code.

Since using a smaller field size is certainly germane to WSN applications, the time consumed for these calculations can be further decreased.

Implementation on the Intel Mote 2 Platform

The methodologies for self-certified key generations developed here were also implemented on the Intel Mote 2 [6] platform (see figure 3-5). This recently offered high-end, low power module

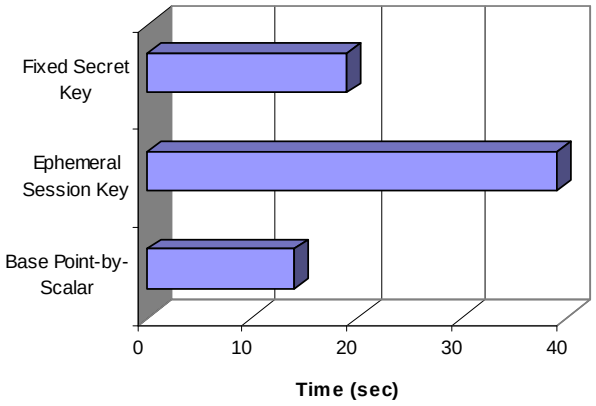

Figure 3-4: Self-certified key generation timing requirements for the TelosB mote

employs the Intel PXA271 XScale Processor running at a clock frequency ranging from 13 MHz to 416 MHz (in contrast to the telosB motes where the only frequency is 4 MHz) The core frequency can be dynamically set in software, allowing the designer to carefully adjust the timing/power trade-off so as to optimize performance of a particular application. The self-certified algorithms (both fixed and ephemeral) were again implemented using functions taken from the TinyECC package [49]. The original code provided by Malan *et al.* [45], which was designed for the 8-bit MICA2 mote [2], was revised and optimized for the Intel Mote 2 implementation. This package provided a basic library of ECC-based functions, including scalar multiplication and exponentiation operations. Customizations for the XScale processor, including 32-bit operation optimizations were carried out. In addition, supplementary functions, such as efficient Montgomery arithmetic were added. Nodes exchanged messages using a 2.4 GHz, IEEE 802.15.4 compliant, low-power radio transceiver.

The ECC key size used in the measurements was 160 bits, the cryptographic complexity of which is equivalent to that of 1024-bit RSA (as indicated above). The results were very encouraging: at a frequency of 312 MHz, the process of scalar-point multiplication required only 45 msec, while consuming only 24 mJ. It is observed that while the time it takes to perform the entire key generation process scales linearly with regard to the clock frequency, the energy does not. A strong advantage was observed for operating at higher frequencies. At

Table 3.3: Time and energy consumption for scalar-point multiplication on the Intel2 sensor platform.

Scalar Point Multiplication			
EccM			
Time (msec)	Voltage (v)	Current (mA)	Energy (mJ)
190	3.8	137	99
TinyEcc			
Time (msec)	Voltage (v)	Current (mA)	Energy (mJ)
42	3.8	137	22
Radio Transmission (including a 7 byte header)			
Time (msec)		Energy (mJ)	
~15		0.127	

a frequency of 312 MHz, for example, fixed-key generation can thus be achieved in less than 50 msec, consuming approximately 25 mJ (including all communication overheads). These surprising results clearly pave the way for broader development of resource-efficient security mechanisms for wireless sensor networks. Moreover, it should be noted that since using a smaller field size is certainly germane to sensor network applications, the time consumed for these calculations can be further reduced.

Table 3.3 describes the time and energy consumption for scalar-point multiplication on the Intel2 sensor platform (using a field size of 163 bits) on a 312 MHz clock. Both implementations on the EccM and TinyEcc are indicated.

Comparison between the two platforms As can be observed form the previous two sections, the performance achieved for point-by-scalar multiplication using the Intel Mote 2 is significantly higher than that of the TelosB motes, in terms of both time and energy consumption. While Malan's code [45] demonstrated very interesting results, a more efficient code was that of the TinyECC package [49]. Table 3.4 summarizes the primary measurements obtained.

To provide a reference point for the computational effort in performing point-by-scalar multiplication on the Intel Mote 2 node, listening over the radio for one second requires approximately 57 mJ. One of the very powerful attributes of the Intel mote 2 relates to the fact its Intel PXA271 XScale processor can operate at a clock frequency ranging from 13 MHz to 416 MHz. The clock frequency can be determined dynamically (i.e. in run-time) in software. This offers great flexibility to the designer in terms of controlling the power consumption at

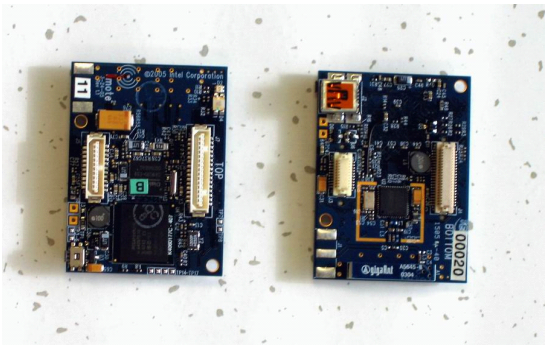

Figure 3-5: The Intel Mote 2 sensor platform

Table 3.4: Point by scalar timing and energy requirements. TinyECC measurements are provided for both TelosB and Intel Mote 2 platforms.

	TinyECC [49]	
	Point-by-scalar multiplication	
	TelosB	**iMote 2**
Computation time	16 s	42 ms
Estimated energy	76 mJ	22 mJ
	Radio transmission	
Computation time	~15 msec	~15 msec
Estimated energy	0.038 mJ	0.127 mJ

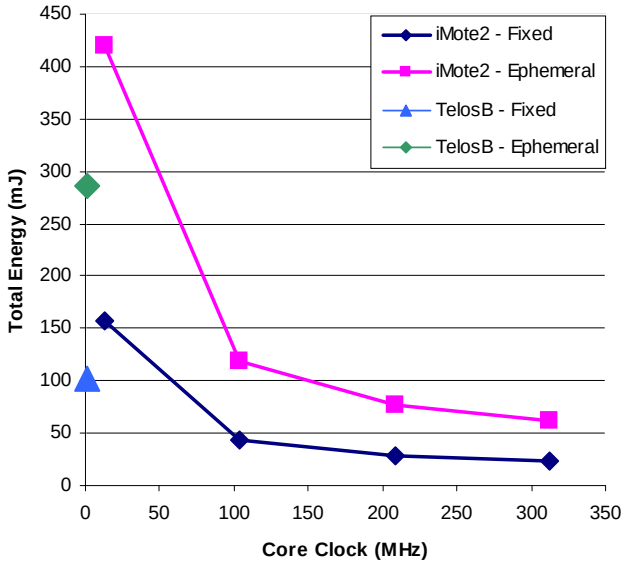

Figure 3-6: Energy consumption for self-certified key generations on TelosB and Intel Mote 2 platforms

any given time. The TelosB, on the other hand, only runs at a frequency of 4 MHz. We can see that the time it takes to calculate a point-by-scalar multiplication using the Intel mote 2 is three orders of a magnitude shorter than the time it takes to calculate a point-by-scalar multiplication using the TelosB (when using the same TinyEcc package). These results are very encouraging and justify the claim that ECC can be implemented on WSNs. Figure 3-6 depicts the total energy consumed while establishing a 160-bit key for both platforms.

The reason for the reduced power consumption with the increase in clock frequency on the Intel mote 2 can be explained as follows. The XScale processor, as any other processor, has fixed peripheral modules that consume constant power. These include timers, interrupt controller, bus arbitration unit, etc. The interrupt controller, for example, continues to operate at a frequency of approximately 4 MHz, regardless of the base CPU frequency selected. To that end, let P_{CPU} denote the fixed power (i.e. frequency independent) consumed by these peripheral processor units. We then let P_b represent the power consumed by the processing of the PKC related functions. If we define $t_{PKC} = \frac{\alpha}{f}$ as the time consumed by the PKC process, where α is a constant and f denotes the clock frequency, then the total energy consumed is

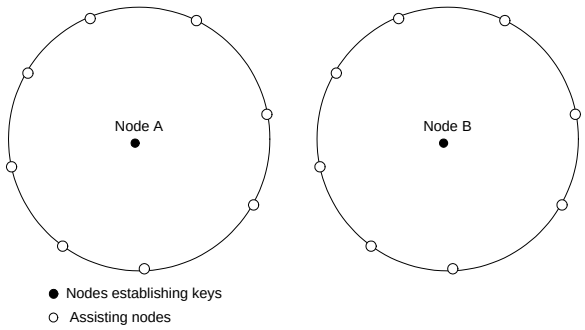

Figure 3-7: A simplified network model demonstrating the efficiency of the offloading approach which affects the network lifetime

given by

$$E_{TOTAL} = E_{CPU} + E_{PKC}(f) = \frac{\alpha}{f} P_{CPU} + P_b, \tag{3.5}$$

which explains the convex curves observed in figure 3-6

3.3.3 Performance Gain Toward Network Lifetime

Next, we consider the implications of the off-loading approach to the overall network lifetime. The latter is measured from the instant the network becomes active until the first node runs out of power. Although the off-loading approach has proven to save computation time and energy consumption for a pair of nodes establishing the shared key, during the off-loading process, extra communication energy is also consumed. Therefore, analyzing the off-loading approach to the overall network performance and network lifetime is an appropriate performance metric. We shall refer to a simplified network model (figure 3-7) to demonstrate the efficiency of the off-loading approach affecting network lifetime.

The model consists of two non-overlapping clusters of nodes, each of which have a cluster head. Let us assume that the two cluster heads exchange keys regularly as means of establishing secure links facilitating the exchange of confidential information. Let us further assume that each of the cluster heads is aided by a (possibly different) node, who is a member of the respective cluster. Since all nodes compute at least as much as the cluster heads, it is apparent that the network lifetime is determined by the lifetime of the head nodes

In order to understand the performance gain on each of the sensor node platforms used,

there is a need to define a few basic parameters:

t_{comp} - the time it takes to perform the calculation of a point-by-scalar multiplication

t_{comm} - the time it takes to transfer or receive the keys

N_1 - the number of keys that can be generated by a single node without off-loading

N_2 - denotes the number of keys that can be generated by a single node with off-loading

E_{comm} - the energy consumed when communicating (i.e. exchanging a packet)

E_{comp} - the energy consumed in computing a point-by-scalar multiplication

E_B - the initial battery energy of each node

In the first case, whereby no off-loading is assumed, we have

$$N_1 = \frac{E_B}{2E_{comp}}, \tag{3.6}$$

since each node performs two exponentiations. For the second case, when off-loading is employed, the number of keys that can be generated is given by

$$N_2 = \frac{E_B}{2E_{comm} + E_{comp}}, \tag{3.7}$$

since each node performs only one exponentiation, but is required to transmit and receive a packet, identical in length to the length of a key.

In order to derive a metric for the network lifetime, we shall assume that on average the application requires that η keys be generated each hour. A reasonable value for η can be, for example, twelve which represents the scenario that on average every five minutes a new key is required. Consequently, the network lifetime for the case of no off-loading is N_1/η while for the case that off-loading is utilized it is N_2/η.

Technical specifications of the TelosB and Intel mote 2 platforms

The TelosB platform is powered by two AA batteries in series operating nominally at 3V, offering approximately $E_B = 9,500\ J$ [27]. Based on the timing and current measurements summarized in table 3.1, for the TelosB platform, the energy consumed in calculating a single

exponentiation and transmitting a key is, respectively,

$$\begin{aligned} E_{comm} &= 19.5 \text{ mAh} \times 3V \times t_{comm} \\ E_{comp} &= 1.8 \text{ mAh} \times 3V \times t_{comp} \end{aligned} \tag{3.8}$$

On the Intel mote 2 platform the same considerations apply, whereby performance gain depends on the processor clock frequency (ranging from 13 MHz to 416 MHz). Here, we have chosen to concentrate on the 312 MHz option, since it yields the lowest total energy consumption. Three AAA batteries were assumed (offering approximately 4000 J) and a voltage of 4.4V is drawn.

Network Lifetime

Figure 3-8 illustrates the expected node lifetime for both TelosB and Intel 2 motes. The assumption is that the cryptographic process consumes 20% of the computational effort involved when an event occurs. As can be seen, as the average interval separating two consecutive key generations grow, the node's lifetime increases as well. Since the network lifetime is defined as the time it takes for the first node to run out of battery, then these figures reveal the network's lifetime as well. Since the energy consumed while calculating a point-by-scalar multiplication is significantly lower in the Intel mote 2 than the energy consumed while calculating a point-by-scalar multiplication on the TelosB motes, the network's life time when using the Intel motes 2 is significantly higher. All calculations here have been done according to the protocol of establishing a self-certified ephemeral key without off-loading (i.e. each node calculates two scalar by point multiplications). While off-loading significantly decreases the time it takes for a node to calculate an ephemeral key, using the off-loading procedure does not improve the network's lifetime since, on average, all nodes would have done the core one multiplication for themselves and assisted with the off-loading for another node in need (increasing the number of multiplications calculated to two). Although the total energy consumed by the off-loading technique is a bit higher due to the communication overhead, such energy consumption is distributed across multiple nodes. This results in a longer network lifetime, since it takes longer, on average, for the first node to run out of energy.

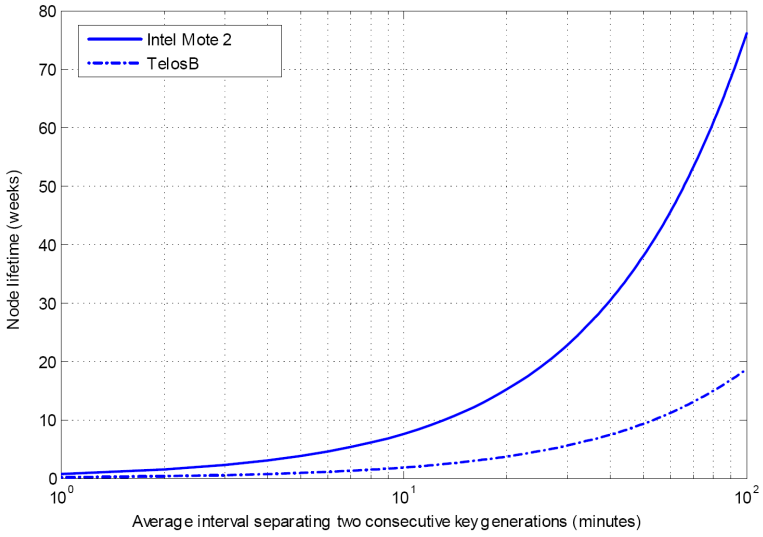

Figure 3-8: Node life time as a function of ephemeral key-generation frequency, assuming 160-bit keys

3.4 Network Lifetime Simulations

In order to obtain a coarse assessment of the impact of off-loading computations on the network lifetime, a Matlab simulation platform was employed. The simulated environment consisted of a $N \times N$ grid in which M nodes were uniformly deployed. Each node has a transmission radius, r_t, and a sensitivity (to event being monitored) radius, r_s. All nodes are assumed to have a battery source with capacity E_{bat} (J). Based on the energy consumption figures described in previous sections, each transmission, reception and computation event reduced the battery energy by their respective amounts. Events occur randomly across the grid, whereby each is assumed to be static for a period of time sufficient for a cluster of nodes to sense it and act accordingly. Each node, upon sensing the event, attempts to establish a key with two other nodes in the cluster, as part of the group key establishment process. Should an assisting node be available (i.e. a node that is not part of the sensing cluster but is close enough to the node requesting assistance), it shares the computational load. It is further assumed that the collaborative signal and information processing that is carried out, following the key establishment phase, requires p times more energy than the key generation did. All nodes that sense the event are included

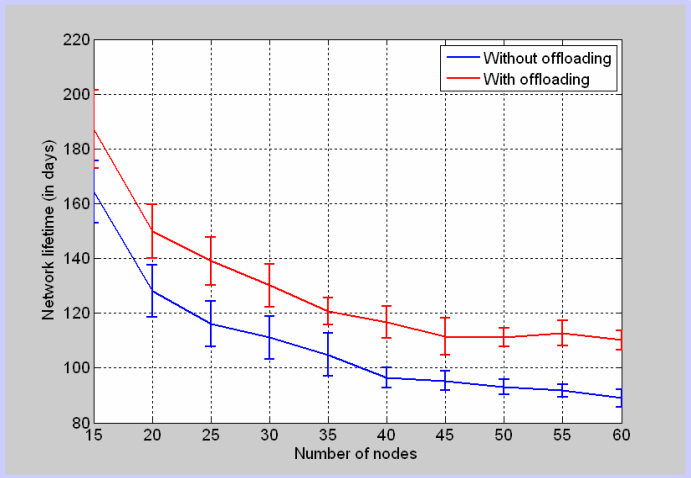

Figure 3-9: Network lifetime as a function of the node density

in the active cluster.

The goal of the simulation is to assess the impact of the off-loading scheme on the overall network lifetime. Each simulation iteration is executed until the first node runs out of battery, so as to be consistent with the definition of a network lifetime. Assuming that on average 30 minutes separate two consecutive events, and that $N = 300, r_s = 80, r_t = 50, Ebat = 2000_J$, figure 3-9, depicts the network lifetime (in days) as a function of the number of nodes (M) deployed. Naturally, as the node density increases, so does the probability that an event will be sensed, thereby incurring energy consumption for both communications and computations. That suggests a negative impact on the network lifetime. However, higher node densities increase the probability that assisting nodes will be found for off-loading computations. This helps distribute the energy-consumption, thus increasing network lifetime.

As can be observed from figure 3-9, the off-loading procedure increases the network's lifetime It is also clear that with or without offloading, for a denser network, the lifetime decreases. This is due to the fact that since the network is more dense, an event is sensed by more nodes (which improves the accuracy of the monitoring process as a whole), and more assisting nodes are called upon. The impact of off-loading results in an increase of about 20% in network lifetime, which is a highly desirable property. In addition to the increase in network lifetime, it is important to note that the main attribute of off-loading is reduction in the time it takes to establish the

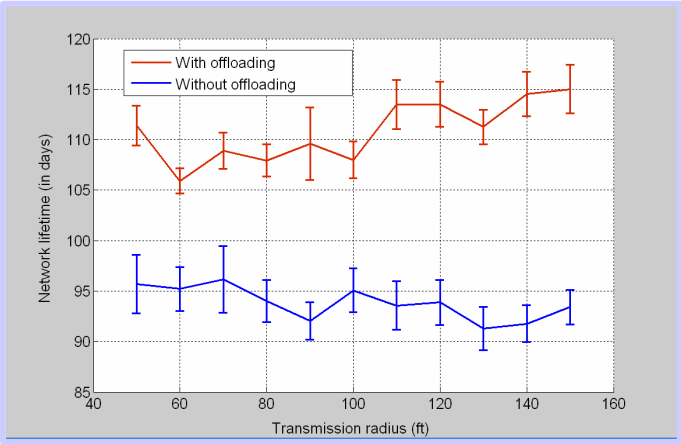

Figure 3-10: Network lifetime as a function of the transmission radius

secret key (since nodes operate in parallel). Moreover, we see that even for a small number of nodes, utilizing the off-loading scheme results in a longer lifetime, primarily due to the fact that any off-loading is better than none.

When the network lifetime was studied as a function of node transmission radius (see figure 3-9), it appeared that beyond a certain transmission range, the network lifetime increased (insignificantly). This can be contributed to the fact that the larger the transmission radius is, the more assisting nodes that can be found. Figure 3-10 depicts the results for the same parameters as in figure 3-9, with $N = 64$. For both figures 3-9 and 3-10, 30 runs were performed in each operation condition, with results reflected by the error graphs.

Chapter 4

Delay-Efficient Group Key Generation

A pivotal theme in this dissertation pertains to *authentication aspects* of key-generation techniques in WSNs, with an emphasis on *efficiency* and *energy preservation*. The group key generation scheme presented in section 3.2.1 is based on pairwise communication and as such carries an inherent drawback of substantial process delay. The latter impacts the time it takes to establish a group key from the instant that the process begins until its completion. This group key generation scheme's final stage of the process, which comprises the exchange and validation of the secret key to be used, introduces latency that is linear with respect to the number of nodes in the cluster. This is true for all schemes that are based on pair-wise exchange of information and prohibits the overall scalability key generation framework. To address this key issue, this chapter describes a more delay-efficient key generation scheme that is much less dependent on the particular topology of the cluster.

We begin by asserting that fully-certified key generation procedures are comprised of the following three generic steps:

1. Verifying the authenticity of users' public keys by validating their associated certificates. Here the validator is assured that a submitted public key corresponds to the ID of the user that claims to be the owner of the key. Such validation is achieved via reference to the CA's public key.

2. Verifying the authenticity of the exchanged ephemeral values. This is based on referring to the users public key, whose authenticity was verified in the preceding step.

3. Generating the session key, based on the ephemeral values verified in Step 2.

Various group-key generation procedures and their associated authentication schemes have been proposed in recent literature ([36], [12], [25], [43]). These works are based on distinctly performing all three steps described. Here, a novel group-key generation is treated, unrelated to the group-key generation procedures presented in the references provided. This method devises a complete group-key generation procedure in which steps 2 and 3 above are joined into one mathematical operation. Step 1 is assumed to be independently performed prior to each cluster formation event. It will be shown that the contribution of this new procedure does not concern savings in computational efforts, but rather ease of management and a substantial reduction in the overall time consumed by the key generation process.

4.1 Introductory Remarks

In order to understand the rationale behind the group key generation procedure presented in this chapter, we must first look at three different procedures. The first is the Burmester and Desmedt (BD) algorithm [13], [14], which treats step 3 alone by establishing a group key. The second is the Menezes-Qu-Vanstone (MQV) [65] key generation procedure, which treats steps 2 and 3 in one algorithm. The third is the Digital Signature Algorithm (DSA) [34] signature generation and verification, pertaining to steps 1 and 2 independently.

All three procedures are performed over the field $GF(p)$, where the private key and the public key of node i (if used) are x_i and $\mathbf{Y}_i$, respectively, where $\mathbf{Y}_i = \mathbf{G}^{x_i} \bmod p$, and $\mathbf{G}$ pertains to a primitive element in the field. All bold notations represent a point on the elliptic curve, assuming one chooses to work under the field $GF(2^n)$. We next review these fundamental procedures as a prelude to the core contribution.

4.1.1 The Burmester and Desmedt (BD) Group-key Generation

The BD group-key generation procedure comprises of the following steps:

a. Each User$_i$, $i = 1, 2, 3, \ldots, n$, generates $\mathbf{Z}_i = \mathbf{G}^{r_i} \bmod p$, for a randomly selected $1 < r_i < p - 1$.

b. The value $\mathbf{Z}_i$ is broadcasted.

c. Each User$_i$ then calculates and broadcasts $\mathbf{V}_i = [(\mathbf{Z}_{i+1})/(\mathbf{Z}_{i-1})]^{r_i} \bmod p$, where the indices are taken modulo n.

e. Key generation: each User$_i$ computes the key

$$\mathbf{K}_i = (\mathbf{Z}_{i-1})^{nr_i} * (\mathbf{V}_i)^{n-1} * (\mathbf{V}_{i+1})^{n-2} * (\mathbf{V}_{i+2})^{n-3} \cdots \mathbf{V}_{i-2}. \tag{4.1}$$

As a result, all users end up with the same joint session-key

$$\mathbf{K} = \mathbf{G}^{r_n r_{n-1} + r_{n-1} r_{n-2} + \ldots + r_2 r_1 + r_1 r_n} \tag{4.2}$$

This is based on observing that

$$\mathbf{V}_i = [(\mathbf{Z}_{i+1})/(\mathbf{Z}_{i-1})]^{r_i} = \mathbf{G}^{r_{i+1} r_i - r_i r_{i-1}} mod p \tag{4.3}$$

and

$$\mathbf{K}_i = (\mathbf{Z}_{i-1})^{nr_i} * (\mathbf{V}_i)^{n-1} * (\mathbf{V}_{i+1})^{n-2} * (\mathbf{V}_{i+2})^{n-3} \cdots \mathbf{V}_{i-2} = \mathbf{G}^w \bmod p, \tag{4.4}$$

where

$$\begin{aligned} w &= nr_i r_{i-1} + (n-1)(r_{i+1} r_i - r_i r_{i-1}) + \\ &\quad (n-2)(r_{i+2} r_{i+1} - r_{i+1} r_i) + \ldots \\ &\quad + (r_{i-1} r_{i-2} - r_{i-2} r_{i-3}) \\ &= r_i r_{i-1} + r_{i+1} r_i + r_{i+2} r_{i+1} + \ldots + r_{i-1} r_{i-2}. \end{aligned} \tag{4.5}$$

The generation of each of the values $\mathbf{Z}_i$, $\mathbf{V}_i$, $\mathbf{K}_i$ requires one modular exponentiation. To explicitly clarify, the long multiplication associated with the generation of $\mathbf{K}_i$ is of computa-

tional complexity equivalent to two modular exponentiations. This is shown in [14]. Since this procedure concerns only step 3 (generating the session-key, based on the ephemeral values verified in Step 2), authenticity of the ephemeral values $\mathbf{Z}_i$ is assumed. That is, $\mathbf{Z}_i$ is provably associated with an identified User_i. The original presentations of the BD scheme did not treat the issue of authenticating $\mathbf{Z}_i$, which is next addressed.

4.1.2 The Menezes-Qu-Vanstone (MQV) Key Generation

The MQV key generation procedure [65], combines step 2 and step 3, without the execution of step 1. Let z_i denote the scalar presentation of the element $\mathbf{Z}_i$. If $\mathbf{Z}_i$ is an element of $GF(p)$ then in practice it can also be considered as a scalar z_i, when needed. If $\mathbf{Z}_i$ is a point on an elliptic curve, i.e. an element in the field $GF(2^n)$, z_i can be the x-coordinate of $\mathbf{Z}_i$. The MQV procedure takes the following steps:

a. User_i and User_j respectively calculate $\mathbf{Z}_i = \mathbf{G}^{r_i} \bmod p$ and $\mathbf{Z}_j = \mathbf{G}^{r_j} \bmod p$ for randomly selected r_i and r_j.

b. User_i sends $\mathbf{Y}_i$ and $\mathbf{Z}_i$ to User_j; User_j sends $\mathbf{Y}_j$ and $\mathbf{Z}_j$ to User_i.

c. User_i and User_j respectively calculate $\mathbf{K}_i = (\mathbf{Z}_j * \mathbf{Y}_j^{z_j})^{(r_i + z_i x_i)} \bmod p$ and $\mathbf{K}_j = (\mathbf{Z}_i * \mathbf{Y}_i^{z_i})^{(r_j + z_j x_j)} \bmod p$, which is their joint session-key. The value of the generated joint session-key is $\mathbf{G}^{(r_i + z_i x_i)(r_j + z_j x_j)} \bmod p$

d. Key confirmation: confirm that $\mathbf{K}_i = \mathbf{K}_j$. Confirming the validity of the exchanged values is based on explicit certification. Here, $\mathbf{Y}_i$ and $\mathbf{Y}_j$ are validated based on executing Step 1 described before.

The advantage of MQV lies in the fact that the validity of the ephemeral values $\mathbf{Z}_i$, $\mathbf{Z}_j$ does not have to be established by itself. Instead, Steps 2 and 3 are combined into a single step.

4.1.3 Digital Signature Algorithm (DSA)

The DSA procedure [34] can be used for signature generation as well as signature verification (which are an inherent part of steps 1 and 2). The following are the two procedures.

DSA signature generation

The signer, User_i, generates a signature based on his knowledge of the private key x_i. Any party is then able to verify User_i's signature by referring to the public key $\mathbf{Y}_i = \mathbf{G}^{x_i} \bmod p$. Here: q is a prime divisor of $(p-1)$. $\mathbf{T} = \mathbf{W}^{((p-1))/q} \bmod p$, for any $1 < \mathbf{W} < (p-1)$ such that $\mathbf{W}^{((p-1))/q} \bmod p > 1$. In the lines of DSA specifications, User_i signs a message m by generating a random $0 < k < p-1$ and calculating

$$\mathbf{L} = (\mathbf{T}^k mod p) \text{mod} q \text{ and } s = (k^{-1}(H(m) + x_i * L)) mod q, \tag{4.6}$$

where $H(m)$ is a hash of m and L is a scalar representation of $\mathbf{L}$. The signature is the pair $\{\mathbf{L}, s\}$, submitted together with m.

DSA signature verification

Let $m', \mathbf{L}'$ and s' denote the received versions of $m, \mathbf{L}$ and s, respectively. To verify the authenticity of m' (i.e., establish the fact that $m\prime = m$), the verifier calculates:

$w = (s')^{-1} \bmod q$

$u1 = H(m') * w \bmod q$

$u2 = L' * w \bmod q$

(*) $\mathbf{V} = (\mathbf{T}^{u1} * \mathbf{Y}_i^{u2} \bmod p) \bmod q$.

If $\mathbf{V} = \mathbf{L}'$ then the signature is verified. That is, this step provides a yes or no answer regarding the validity of m.

4.2 The Combined BD-MQV Group Key Generation

The procedure presented next is proposed to address latency in group-key generation for WSNs. It concerns joining Steps 2 and 3 within a group-key generation framework. The suggested procedure takes the following steps:

a. User_i, $i = 1, 2, 3, \ldots, n$, generates $\mathbf{Z}_i = \mathbf{G}^{r_i} \bmod p$, for a randomly selected $1 < r_i < p-1$.

b. Each User_i broadcasts $\mathbf{Y}_i$ and $\mathbf{Z}_i$.

c. Each User$_i$ calculates and broadcasts

$$\mathbf{V}_i = [(\mathbf{Z}_{i+1}\mathbf{Y}_{i+1}{}^{z_{i+1}})/(\mathbf{Z}_{i-1}\mathbf{Y}_{i-1}{}^{z_{i-1}})]^{(r_i+z_ix_i)} \bmod p, \tag{4.7}$$

where, as before, x_i and $\mathbf{Y}_i = \mathbf{G}^{x_i} \bmod p$ are the private and public keys of User$_i$.

d. Key generation: User$_i$ computes the key

$$\mathbf{K}_i = (\mathbf{Z}_{i-1}\mathbf{Y}_{i-1}{}^{z_{i-1}})^{n(r_i+z_ix_i)} * (\mathbf{V}_i)^{n-1} * (\mathbf{V}_{i+1})^{n-2} * (\mathbf{V}_{i+2})^{n-3} ... \mathbf{V}_{i-2}. \tag{4.8}$$

All users end up with the same key, having the form $\mathbf{G}^v \bmod p$, where

$v = n(r_{i-1} + x_{i-1}z_{i-1})(r_i + z_ix_i) +$

$+(n-1)(r_i + z_ix_i)(r_{i+1} + x_{i+1}z_{i+1} - r_{i-1} - x_{i-1}z_{i-1}) +$

$+ \ (n-2)(r_{i+1} + z_{i+1}x_{i+1})(r_{i+2} + x_{i+2}z_{i+2} - r_i - x_iz_i) + ...$

$+(r_{i-2} + z_{i-2}x_{i-2})(r_{i-1} + x_{i-1}z_{i-1} - r_{i-3} - x_{i-3}z_{i-3})$

$= (r_n + z_nx_n)(r_{n-1} + z_{n-1}x_{n-1}) +$

$+ \ (r_{n-1} + z_{n-1}x_{n-1})(r_{n-2} + z_{n-2}x_{n-2}) + \ldots$

$+(r_1 + z_1x_1)(r_n + z_nx_n) \bmod (p-1)$

For $n = 2$, $\mathbf{K} = \mathbf{G}^{2(r_1+z_1x_1)(r_2+z_2x_2)} \bmod p$, which is the (squared) MQV key, with similar security considerations

e. Key confirmation: We note that if User$_i$ does not know $\log \mathbf{Y}_i$ (i.e., it does not know x_i such that $\mathbf{Y}_i = \mathbf{G}^{x_i} \bmod p$), the chained procedure under which all $\mathbf{K}_i$ end up the same fails, and hence each user ends up with a different key. Confirming that the users share the same key proves that all users know and have used the log of their certified public keys, which is the essence of discrete-log-based authentication.

Similar to the general MQV case, the procedure presented above is 'semi-self-certified'. That is, steps 2-3 are combined into one step, but unlike fully self-certified procedure, step 1 is still performed independently.

A note on group-wise key-confirmation: Step (e) above concerns key-confirmation, whereby all parties of the group are to be convinced that they share the same key. In practice, an implicit key confirmation is recommended in such cases. That is, the group members skip step (e) and start communicating using their shared key. Only parties that have the correct key will be able to correctly encrypt/decrypt messages, and there is no need for explicit preliminary key confirmation step.

The calculation of $\mathbf{V}_i$ in this combined BD-MQV procedure requires three exponentiations, compared to one exponentiation in the original BD procedure. Since in the latter there is also a need to verify the authenticity of the exchanged ephemeral values, a comparison with the number of exponentiations in the DSA procedure is relevant. These extra two exponentiations are equivalent in complexity to a discrete-log-based signature verification, as described in subsection 4.1.3 by the marked (*) in the DSA verification procedure. The calculation of $\mathbf{K}_i$ requires two exponentiations, as in the original BD procedure. Since in the latter, there is also a need to verify the authenticity of the exchanged ephemeral values, we must also take into account the extra exponentiation in the signature generation operation as described in subsection 4.1.3 by eq. (4.6) in the DSA generation procedure. Altogether, executing the combined BD-MQV procedure requires five exponentiations, whereas in the original BD procedure along with the needed DSA procedure there are six.

Executing an independent signature generation/verification procedure performed when validating ephemeral values (like in the DSA procedure) is not necessary in this group key generation. This also introduces significant savings in management overhead. The original BD scheme does not treat the need to validate the authenticity of the transmitted $\mathbf{Z}_i$, as described in step 3. This overhead is omitted in the proposed procedure. However, as in the case of the original BD, the procedure necessitates the computational overhead of executing step 1

We next compare the different performance metrics of the pairwise key generation scheme described in chapter 3 with the combined BD-MQV scheme, as summarized in table 4.1. In the case of operating over the field of $GF(2^n)$, every exponentiation is substituted by a point-by-scalar multiplication. When examining the procedure one notes that there are six multiplications involved: one for step a, three for step c and two for step d. When adding the use of DSA for verifying the authenticity of users' public keys, the signature verification step necessitates

two extra multiplications. Altogether, each node is required to perform eight multiplications. When comparing this procedure to the ephemeral procedure presented in chapter 3, we notice that in the ephemeral self-certified method there are only six multiplications involved.

The next criterion to be examined is the total number of value transmissions and receptions involved in each protocol. The pairwise-based scheme requires each node to transmit three values to its neighbors in the ring topology. Correspondingly, each node receives 6 values from its neighbors (while three of them are recieved at the time of the other node's transmission), bringing the total number of values exchanged thus far to 6. This pertains to the key establishment phase, however another message is received and transmitted by each node during the propagation of the secret key. This brings the total number of values exchanged in the network to $8N$. In the combined BD-MQV scheme, each node performs 3 transmissions, corresponding to $\mathbf{Y}_i$ and $\mathbf{Z}_i$ and $\mathbf{V}_i$, and receives $3(N-1)$ values. Given that the BD-MQV protocol is broadcast-based, the total number of transmissions is $3N$.

When comparing the total process latency, however, there are significant differences between the two approaches. Let t_{mult} and t_{comm} denote the time consumed by a multiplication and either a value transmission or reception, respectively. As can be seen in table 4.1, neglecting the time consumed by the multiplications, which are independent of the network size, the latency in the case of BD-MQV is 37.5% (3/8) of that of the pairwise scheme. This is a substantial gain in terms of the overall process delay.

Naturally, expediting the process comes at a cost; in our case, energy cost. We define e_{mult} and e_{comm} as the energy consumed by a single multiplication and a single transmission/reception event, respectively. While the pairwise scheme requires $(11N)\, e_{comm} + (4N)\, e_{mult}$, each node in the BD-MQV case transmits 3 values and receives $3(N-1)$ values (from the other $N-1$ nodes in the network), which translates to $3N^2$ overall message exchanges. This is indeed a considerable energy cost to pay for a speedup gain, however in some applications such a trade-off may be viewed as acceptable.

Table 4.1: Performance comparison between the pairwise key generation scheme and the combined BD-MQV method.

Performance metric	*Pairwise*	*Combined BD-MQV*
Multiplications	$4N + 2N$ (*a priori*)	$6N + 2N$ (*a priori*)
Transmission/receptions time	$(8N)\, t_{comm}$	$(3N)\, t_{comm}$
Total energy	$(11N)\, e_{comm} + (4N)\, e_{mult}$	$\left(3N^2\right) e_{comm} + (6N)\, e_{mult}$
Process total latency	$(8N)\, t_{comm} + 4t_{mult}$	$(3N)\, t_{comm} + 6t_{mult}$

4.3 Network Lifetime Simulations

As indicated above, each of the N nodes in a cluster is required to perform $3N$ modular exponentiations as well as receive $N - 1$ messages from other nodes that have broadcasted their respective information. The simulation setup consisted of a 1000 ft. $\times$ 1000 ft. virtual area over which 300 nodes were randomly deployed. These nodes have a sensing radius that ranges between 50 to 300 ft. We further assume that each node has 5000 J of battery energy, which is approximately the capacity of a AAA battery. In an effort to obtain a coarse network lifetime indication, two scenarios were considered: in the first, events occur randomly across the area of interest; in the second, events were generated along random linear trajectories, representing, for example, a path of vehicle motion across a field of interest. Figures 4-1 and 4-2 illustrate a deployment setup of 300 nodes with linear trajectories and randomly occurring events, respectively.

Figure 4-3 depicts the network lifetime estimate as a function of the sensing radius. Each point on the graph is an average over 1000 trials. Events are assumed to occur on average (in both scenarios) once every hour. Up to 20 nodes are assumed to be included in a cluster that is monitoring an event of interest. As such, if an event is sensed by more than 20 nodes, the rest are assumed inactive in the context of that particular event. Moreover, the data security portion is assumed to constitute 10% of the overall code running on the node. This appears reasonable given the complexity of current WSN applications. We note from the results that as the sensing radius increases, the network lifetime reaches an asymptotic value in both scenarios. This can be explained by the fact that as the sensing radius increases so does the probability of a node participating in the cluster that forms around an event. As a result, it will take a shorter amount of time for one of the nodes to run out of energy. It is also noted that when events

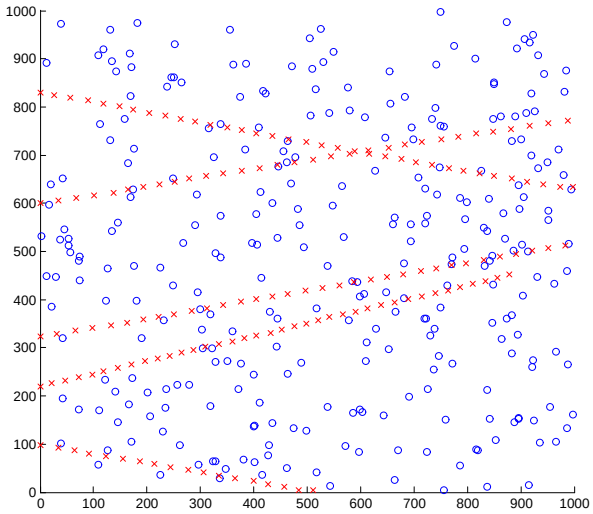

Figure 4-1: An illustration of a 1000 ft $\times$ 1000 ft area with 300 randomly deployed nodes (circles) and linear trajectories of events ('x' symbols).

are correlated in time, as is the case with the second scenario, the network lifetime decreases. This can be intuitively appreciated since individual nodes have a higher probability of being activated several consecutive times as they overlap with the path of a given trajectory, thus concentrating the energy consumption on a smaller set of nodes yielding a lower overall network lifetime.

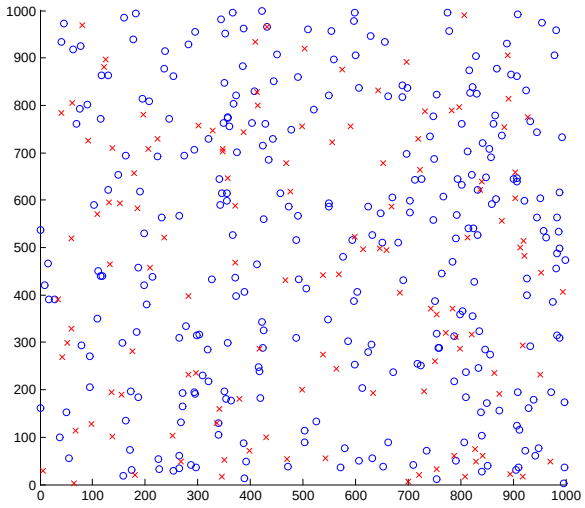

Figure 4-2: An illustration of a 1000 ft $\times$ 1000 ft area with 300 randomly deployed nodes (circles) and random events ('x' symbols)

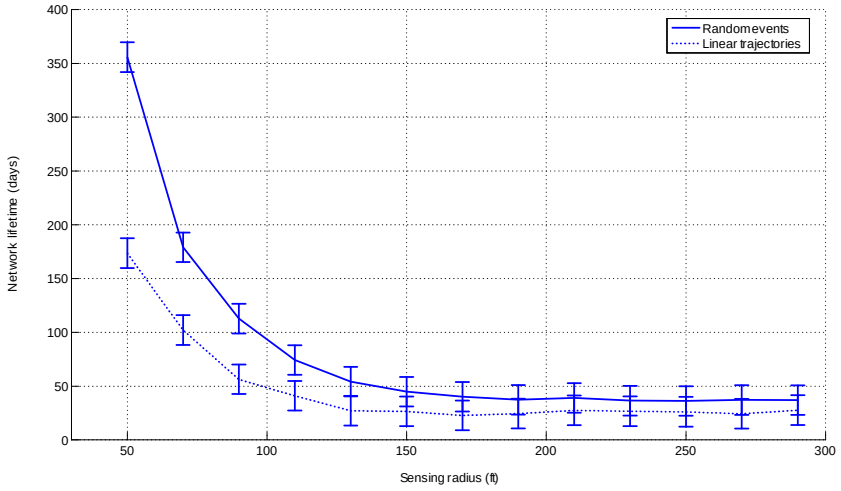

Figure 4-3: Network lifetime for the BD-MQV method as a function of the sensing range. The security portion of the code is assumed to be 10% of the overall computational load.

Chapter 5

Countering Denial of Service (DoS) Attacks

A fundamental requisite for security, other than providing data confidentiality and authentication, is Denial of Service (DoS) mitigation. However, the computational effort involved in performing PKC operations remains substantial. From an energy consumption perspective, it is imperative that the processing and communication resources be utilized only when required. To that end, PKC implementations are more vulnerable to Denial of Service (DoS) attacks when compared to traditional security methods that require less resources. In particular, if a malicious party attacks a sensor node by repetitive requests to establish a key, the resources of the attacked node can be exhausted quite rapidly. Combatting DoS attacks is the last frontier to be conquered prior to making PKC deployment standard security practice in sensor networks.

This chapter focuses on a public key cryptographic approach for mitigating the impact of DoS attacks in WSNs. The proposed novel RSA-based framework for combating DoS attacks ensures that the malicious party will exhaust its resources prior to exhausting those of its counterparts. In particular, the computational asymmetry in RSA signature generation schemes is exploited to yield a resource-efficient authentication mechanism which helps overcome DoS attacks. Three methodologies for establishing an ephemeral key are presented in this context, in which the proposed DoS mitigation mechanism is an embedded component. Implementation results on the Intel Mote 2 platform substantiate the clear advantages of the proposed method.

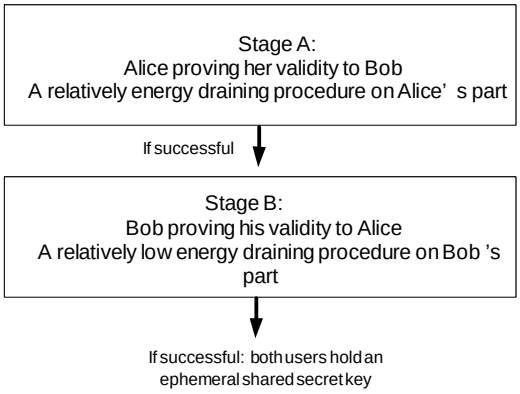

Figure 5-1: The proposed procedure for Denial of Service (DoS) prevention and ephemeral key-generation.

5.1 Outline of the Proposed DoS Mitigation Procedure

The procedures for key generation described in sections 2.5.2 and 2.5.3 do not include any mechanism for DoS mitigation. The DoS attack considered would easily occur when a malicious node repeatedly approaches legitimate nodes requesting the establishment of a joint secret key. The energy consumed by the legitimate nodes in the process of key generation is substantial. Therefore, such an attack strategy can drain their energy. An efficient DoS mechanism should be able to prevent such attacks. The proposed DoS mitigation approach has two complementing parts. The first pertains to the instigator, Alice, who has to prove her validity to Bob, the party (node) approached. The second part, which takes effect only if Alice has indeed proven her validity, pertains to Bob, who is required to prove his validity to Alice. We will demonstrate that if the two procedures are successful, i.e., the identities of both Alice and Bob are validated, then an ephemeral key can be issued. The latter implies that each time a certain legitimate node wishes to establish a key with a neighboring node, not only is a DoS attack prevented, but a different secret key will be generated. Figure 5-1 provides an illustration of the proposed framework.

We shall refer to the following notations in the context of the proposed DoS mitigation scheme:

- $n_i \longrightarrow$ user i's public key

- $d_i \longrightarrow$ user i's private key
- $CR_i \longrightarrow$ user i's (CA issued) certificate
- $ID_i \longrightarrow$ user i's public key identification

Notice that in sections 2.5.2 and 2.5.3, where ECC based self-certified keys were established, the private key, x_i, was a scalar and the public key, $\mathbf{U}_i$, was a point on the elliptic curve, whereas in the scenario of DoS mitigation depicted here, both n_i and d_i are scalars of the same length. The latter are RSA related parameters.

The following sections describe, in detail, the two stages of the DoS mitigation method.

5.1.1 The Instigator Node Proving Its Validity

The specific scenario described in this case pertains to a malicious node that is attempting to drain the energy of a trusted nodes. The first step of a key establishment protocol consists of an instigator node (Alice) initiating communications with another node (Bob). We shall refer to the instigating node as a suspicious node that is required to prove its identity. We thus expect that during the first stage of the key exchange process, the majority of the energy consumed will be on Alice's part. This would mean that if a DoS attack is carried out, whereby a malicious node repeatedly attempts to generate a key with a valid node, the latter will be required to use as little energy as possible. We must assume that most of the nodes are not jeopardized; hence the instigating nodes are to be presumed innocent until proven guilty. In other words, the amount of energy drained from Alice will be significant, yet not too high so as to not deplete her battery too fast. However, if Alice is malicious, and attempts to establish keys with various nodes, she will eventually run out of energy and /or expose her malicious nature.

The method described next is based on the notion of key transport [29] using RSA [60] with $e = 3$. We note that $e = 3$ is considered sufficiently secure [35]. The following four steps constitute an ephemeral key exchange procedure that embeds the DoS mitigation mechanism:

Step 1 - Alice sends Bob her public key, n_A, her identification, ID_A, and her certificate (issued by the CA), CR_A. The certificate is the *CA's signature on the association* between n_A and ID_A. An example for such an association can be: $n_A \oplus ID_A \equiv H(n_A, ID_A)$. Note that ID_A

can be a small number; n_A can be 1024 bits (as in the protocol used here), hence $H(n_A, ID_A)$ depends on the length of n_A. In this case, $CR_A = [H(n_A, ID_A)]^{d_{CA}} \bmod n_{CA}$. Naturally, only the CA can create the CR_A by using its private key d_{CA}.

Step 2 - Bob verifies the validity of the certificate (CR_A) by testing the equality $(CR_A)^e \bmod n_{CA} \stackrel{?}{=} H(n_A, ID_A)$. If the latter holds, Bob knows that n_A and ID_A are undeniably connected. Since $e = 3$, this step requires Bob to compute **only two** modular multiplications [48]. If indeed $(CR_A)^3 \bmod n_{CA} = H(n_A, ID_A)$, Bob can then continue with generating a message m (it will later be shown how this message is utilized as part of the key generation process), compute $t = m^e \bmod n_A$ and transmit t to Alice. Again, since $e = 3$, Bob has to calculate **only 2** modular multiplications at this step.

Step 3 - Alice needs to prove that she indeed possesses the private key d_A, proving to her counterpart that her identity is valid. This is true since the CA would have given this private key only to her. Let s_x denote the number of bits in x, the least significant section of m. Alice needs to calculate $t^{d_A} \bmod n_A = m$ and send Bob x. Message m is comprised out of n bits such that $n >> s_x$. The rest of the bits in the message will be used for the ephemeral key generation, as will later be described.

It should be noted that, in contrast to Bob, Alice has to perform a computationally heavy task as d_A typically consists of either 512 or 1024 bits. To that end, the approach proposed shifts the computational burden on the instigating node.

Step 4 - Bob compares the binary vector x he receives from Alice with the s_x least significant bits in m. If these are identical he determines that Alice's identity is valid. If not, he asserts that Alice is malicious and terminates the key establishment process. In this case Bob performed merely four modular multiplications, two receptions and 1 transmission.

The above process has achieved several key goals. First, the instigating node (Alice) uses more energy than the approached node (Bob) as she calculates $t^{d_A} \bmod n_A$. Yet this is an accepted burden under the assumption that the calculation of $t^{d_A} \bmod n_A$ is performed only once per key generation. Second, if Alice is malicious and attempts to instigate key generation with more than one node, calculating $t^{d_A} \bmod n_A$ for various types of t's (different from one correspondent to another) will drain her energy. Third, if the same ID_A is used over and over again then she is bound to be ignored. If Alice is trustworthy, she will need to use her ID_A

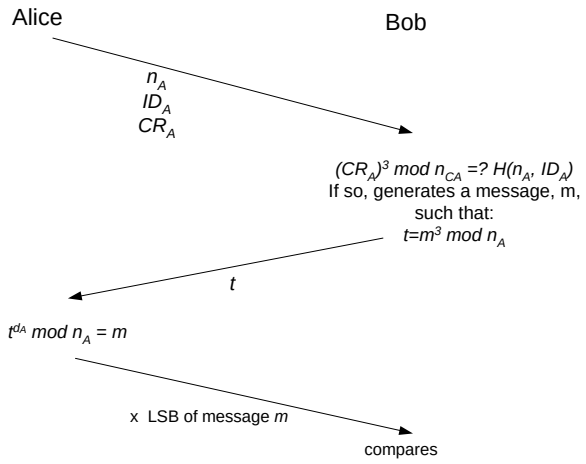

Figure 5-2: DoS mitigation based on the Key Transport procedure.

only twice for both key generations performed (assuming the use of the group key generation described in [50]). Finally, if Alice attempts to impersonate another user by using a different ID_i, then it will immediately be identified since $(CR_A)^e \bmod n_{CA} = H(n_A, ID_i)$ will not hold. In this case, Bob will only have wasted two modular multiplications and one reception. Figure 5-2 illustrates the complete DoS mitigation procedure depicted in stage A.

Two threat models should be considered in this context. First, Alice can attempt to drain Bob's energy by continuously requesting to establish a key, each time using a different ID. Since Bob is only required to calculate $(CR_A)^3 \bmod n_{CA}$ and compare it with $H(n_A, ID_A)$, the computations involved are two Montgomery multiplications alone [48]. Hence the energy consumed in each attempt is relatively small. Moreover, the time Bob spends performing the computations is rather small, thereby not introducing a significant burden in that sense. Second, a malicious node impersonating Alice can repeatedly initiate a key establishment process using ID_A. The question is how can Bob know which messages should be ignored? A possible solution would be to maintain a list of IDs of recent nodes that resulted in failed validation (step 2). Bob will then refrain from proceeding with key generation requests originating from these nodes. A time-out mechanism should be employed such that banning of nodes expires after a reasonable duration of time. An underlying assumption in all threat models considered is that the attacking node has energy resources that are comparable to those of the node attacked.

5.1.2 The Approached Node Proving Its Validity

If the first part of the procedure is successful and Alice has proven that she is who she claims to be, then Bob will need to do the same. However, if the first stage does not pass, Bob assumes that Alice is not valid, and he will discard the rest of the procedure.

The second stage can be realized in three different ways: (1) using self-certified fixed-key generation [50], [8], [7],.(2) using key transport, and (3) using the Elliptic Curve Digital Signature Algorithm (ECDSA) [41]. We next describe each of these methods and discuss their respective advantages and disadvantages. Moreover, it will be shown that in each of the cases an ephemeral key is established, which is a primary goal.

Self-Certified DH Fixed Key-Generation

One of the methods in which Bob can prove his validity to Alice is by using the self-certified fixed-key method described in Subsection 2.5.2. The ephemeral method (described in Subsection 2.5.3) can certainly be used, but when the primary focus is to minimize energy drainage, a self-certified fixed-key generation is advisable (see subsection 2.5.2) since it consists of fewer computations.

As a reminder, the notations are the following: $\mathbf{G}$ - a generating group-point, used by all relevant nodes; $ord\mathbf{G}$ - the order of $\mathbf{G}$. (exponents are calculated *modulo* $ord\mathbf{G}$); d - the CA's private key; $\mathbf{R}$ - the CA's public key (where $\mathbf{R} = d \times \mathbf{G}$); x_i - the *private* key of node i served by the CA; $\mathbf{U}_i$ - the *public* key of a node i served by the CA; ID_i - the identification details, or attributes, of node i; $H(v, \mathbf{W})$ - a scalar obtained by performing a hash transformation on the scalar v and group point $\mathbf{W}$; h_i - a random 160-bit scalar generated by the CA (for the purpose of calculating x_i); N_i, N_j- sensor nodes i and j, respectively.

We now go back to the description of the self-certified fixed-key method used in 2.5.2:

A self-certified DH fixed-key generation is achieved by the following two steps [8] : (1) N_i and N_j exchange the pairs $(ID_i, \mathbf{U}_i)$ and $(ID_j, \mathbf{U}_j)$, respectively, and (2) N_i and N_j generate the session-key,

$$K_{ij} \text{ (generated by } N_i) = x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}]$$
$$K_{ji} \text{ (generated by } N_j) = x_j \times [H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}]. \quad (5.1)$$

The two keys are expected to be identical, having the value $x_i \times x_j \times \mathbf{G}$. (i.e., N_i calculates: $x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] = x_i \times [H(ID_j, \mathbf{U}_j) \times h_i \times \mathbf{G} + d \times \mathbf{G}] = x_i \times [H(ID_j, \mathbf{U}_j) \times h_i + d] \times \mathbf{G} = x_i \times x_j \times \mathbf{G}$. Similar logic is applied by the calculations performed at N_j. However, these identities hold only for valid ID's. Therefore, to complete the authentication cycle there is a need for key-confirmation during which the two nodes either verify that they share an identical key by encrypting and decrypting a test value or by establishing a communication session and implicitly verify that they share the same key. Verifying that the keys generated by the two nodes are equal then establishes their correct identities.

A primary contribution offered by this method of self-certified fixed-key generation lies in the number of exponentiations needed to calculate the value $x_i \times x_j \times \mathbf{G}$. As indicated above, each node (among each pair of nodes) calculates the value $x_i \times x_j \times \mathbf{G}$. Note that the calculations performed by N_i are $K_{ij} = x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] = x_i \times H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + x_i \mathbf{R}$. Further note that the calculations have been separated into two parts. The first is a dynamic scalar by point multiplication executed in an ad hoc manner (as it contains the value $\mathbf{U}_j$). The second is a scalar by point multiplication that can be calculated and stored "before" the key-generation session commences, thereby avoiding the need for a real-time exponentiation (as it contains information known *a priori* by node i). It is clear that N_i is able to calculate its session key by a single online exponentiation ($x_i \times H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j$) instead of two. Similar considerations apply to N_j.

We shall refer to the joint fixed-key shared by Alice and Bob as $K_{AB-temp}$. In addition, as an integrated part of the key generation process, if the two generated keys are indeed identical, authentication is achieved. Therefore, the approached node has proven its validity to the instigator.

The goal of the entire procedure is to establish a shared joint secret key. It is highly desirable for that key to be ephemeral, i.e., two nodes generate a different key for each session

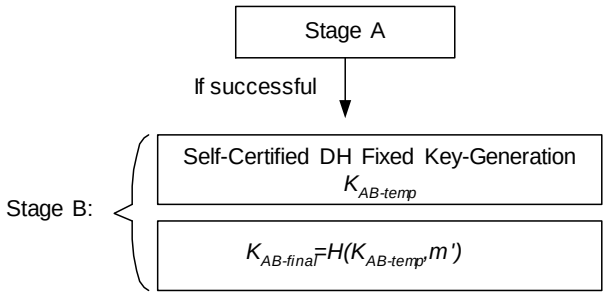

Figure 5-3: Ephemeral key generation and denial of service mitigation using a self-certified DH fixed key-generation.

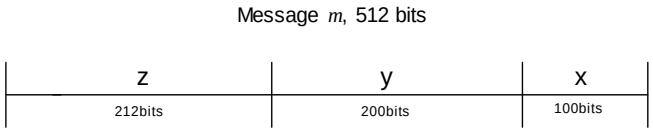

Figure 5-4: Depicting a scenario where the original message is 512 bits.

established. Ephemeral key-generation is more secure and is generally preferred when time and resources permit. A self-certified DH ephemeral key-generation is also possible ([8]), but would consume three times more energy when compared to the fixed-key case. In order to establish an ephemeral key, the two nodes can utilize bits in message m, (generated by Bob) excluding the first x least significant bits. Hence, the final shared ephemeral key can be defined as

$$K_{AB-final} = H(K_{AB-temp}, m'), \tag{5.2}$$

where H is a hash function and m' is the random message m, excluding the x least significant bits (see figure 5-4). Another option would be to simply use the remanding portion of message m, i.e., $m\prime$ as the final ephemeral key.

Key Transport

Bob can validate himself to Alice by using the RSA key transport method, similar to that described in section 5.1.1. The random message m generated by Bob was encrypted using Alice's public key n_{CA} and e. After sending the encrypted message t, such that $t = m^e \bmod n_A$,

Alice can decrypt the message back using her private key, d_A. Eventually, both nodes share the same secret message m. The remaining bits of message m (excluding the s_x least significant bits that were used in stage A) are utilized to establish an ephemeral key. For example, if the length of m is 512 and $s_x = 100$, then there are 412 bits that can be used for authenticating Bob and establishing the ephemeral secret key. In this scenario, y will denote the 200 bits that follow x (as depicted in figure 5-4). The subsequent 212 bits of message m will be labeled z.

The following summarizes the key transport procedure considered:

Step 1 - Bob calculates $S_B = y^{d_B} \bmod n_B$, where y is the next LSB portion of message m.

Step 2 - Bob sends Alice his public key, n_B, his identification, ID_B, his certificate (issued by the CA), CR_B, and S_B. As described above, the certificate is the *CA's signature on the association* between n_B and ID_B. As such, $CR_B = [H(n_B, ID_B)]^{d_{CA}} \bmod n_{CA}$. Only the CA can create CR_B by using its private key d_{CA}.

Step 3 - Alice verifies the following: $(CR_B)^e \bmod n_{CA} \stackrel{?}{=} H(n_B, ID_B)$. If true, Alice knows that n_B and ID_B are undeniably linked. Since $e = 3$, Alice computes only two modular multiplications. To check the validity of Bob, Alice checks the following two equalities

$$(CR_B)^e \bmod n_{CA} \stackrel{?}{=} H(n_B, ID_B) \tag{5.3}$$

$$(S_B)^e \bmod n_B \stackrel{?}{=} y \tag{5.4}$$

If true, Alice knows that the corresponding node is indeed Bob, since only he has the same data, y. The ephemeral key resulting will be denoted by $K_{AB-final} = z$, corresponding to the most significant portion of message m. Figure 5-5 illustrates the complete process.

Elliptic Curve Digital Signature Algorithm (ECDSA)

Bob can also validate himself to Alice by using ECDSA [4]. The latter is a method for digital signatures based on ECC. The ECDSA variation proposed utilizing the components of the message exchanged, m, is:

Step 1 - Bob generates a random number, u, calculates a public value, a point on the curve $\mathbf{V} = u \cdot \mathbf{G}$, where $\mathbf{G}$ is a generating group-point and calculates C, the scalar representation of point $\mathbf{V}$. Next, he computes $L = u^{-1}(y + d_B \cdot C) \bmod ord\mathbf{G}$. Finally, he transmits Alice the

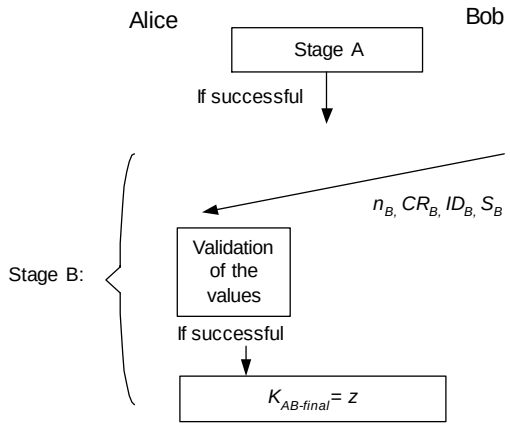

Figure 5-5: Ephemeral key generation and denial of service prevention using key transport.

signature pair (C, L).

Step 2 - Alice calculates $h = L^{-1} \bmod ord\mathbf{G}$, $q_1 = y \cdot h \bmod ord\mathbf{G}$, and $q_2 = C \cdot h \bmod ord\mathbf{G}$. She next obtains the curve point: $\mathbf{P} = q_1 \cdot \mathbf{G} + q_2 \cdot \mathbf{n}_B$, where $\mathbf{n}_B$ is Bob's public key, and calculates C', the scalar representation of point P. The algorithm concludes when Alice validates that $C = C$'. If the latter holds, Bob is validated.

Step 3 - The ephemeral key resulting will be denoted by $K_{AB-final} = z$, corresponding to the most significant portion of message m. Figure 5-6 illustrates the complete process.

5.1.3 Mathematical Considerations

Let us look at a few mathematical calculations pertaining to Subsection 5.1.1. The following calculations are associated with the public and private key possessed by Alice and the CA. (The same calculations will hold to any other node in the cluster):

Calculating the public key n_A and n_{CA} :

We define

$$n_A = p_{1*} p_2$$

$$n_{CA} = p_{3*} p_4$$

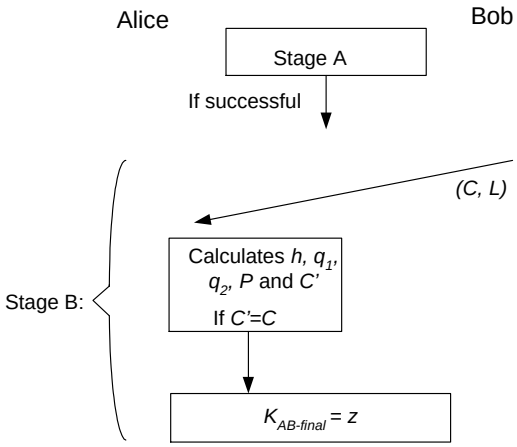

Figure 5-6: Ephemeral key generation and denial of service prevention using ECDSA.

where p_i are Pseudo-random prime numbers of 256 bits. We note that there is a small finite group of 256-bit numbers, hence the use of *pseudo-random prime.* In general, if $a^{p-1} \bmod p = 1$, or: $a^{\frac{p-1}{2}} \bmod p = \pm 1$, for a large pool of different a's, then there is a high probability that p is prime.

The calculation of $a^{\frac{p-1}{2}} \bmod p$ involves the calculation of a modular multiplication of two scalars, as treated by the **Montgomery modular multiplication** procedure

Montgomery Modular Multiplication Procedure:

Lets look at an example where the scalars are x and y. The purpose of the procedure is to yield the result $x \cdot y \ mod \ p$. The procedure processes values which consist of m-bit coefficients. Let v_i denote the i^{th} coefficient of an integer v, where the least significant coefficient is denoted by v_0. We let k be the number of m-bit coefficients in x and y. Moreover, we let r denote the value of $-p_0^{-1} \bmod 2^m$ (To clarify, p_0 represents the least significant coefficient of the modulus p). The Montgomery procedure can thus be described as follows:

begin

$s = 0, t = 0, v = 0$

for $i = 0$ to $k - 1$

$t = s + x_i \cdot y$

$u = (t_0 \cdot r)_0$

$v = t + u \cdot p$

$s = \frac{v}{2^m}$

end (for loop)

if $s > n$ then $s = s - p$

end

Notice that for each p, we need to calculate the value $r = -p_0^{-1} \bmod 2^m$. This value depends only on the least significant coefficient of the modulus p.

Calculating the private key d_A and d_{CA}:

$$d_A = e^{-1} \bmod \varphi(n_A)$$

- $\varphi(n_A)$-Euler's Totient Function

Euler's Totient Function $\varphi(m)$ returns the number of integers less than m, including 1 that are relatively prime to m. [For $m = p$ (p- prime), $\varphi(p) = p - 1$].

Hence, $\varphi(n_A) = (p_1 - 1)(p_2 - 1)$.

- When using $e = 3$, as in this case, $d_A = e^{-1} \bmod \varphi(n_A) = 3^{-1} \bmod (p_1 - 1)(p_2 - 1)$.

In order to calculate the value $d_A = e^{-1} \bmod \varphi(n_A)$, the following simple procedure can be performed:

Lets choose p_1, p_2 such that p_1, $p_2 = 2 \bmod 3$

$\Longrightarrow (p_1 - 1) \bmod 3 = (p_2 - 1) \bmod 3 = 1$

$\Longrightarrow (p_1 - 1)(p_2 - 1) \bmod 3 = 1$

$\Longrightarrow \varphi(n_A) \bmod 3 = 1$

$\Longrightarrow 2\varphi(n_A) + 1 = 3x, \ \forall x \quad \Longleftrightarrow \quad x = 3^{-1} \bmod \varphi(n_A) \equiv d_A$

$\Longrightarrow$

$$d_A = \frac{2\varphi(n_A) + 1}{3}$$

Using the same logic, the private key of the CA is:

$$d_{CA} = e^{-1} \bmod \varphi(n_{CA}) = \frac{2\varphi(n_{CA}) + 1}{3}$$

(In this case we do not need to evaluate the multiplicative inverse, enabling an easier calculation).

Checking The Certificate:

As described in Subsection 5.1.1, as a part of the procedure, the approached node needs to check the certificate of the instigating node by performing the following:

$CR_A = [H(n_A, ID_A)]^{d_{CA}} \bmod n_{CA}$

As a consequence, the calculations are as follows:

$(CR_A)^e \bmod n_{CA} \stackrel{?}{=} H(n_A, ID_A) \iff (CR_A)^3 \bmod n_{CA} \stackrel{?}{=} H(n_A, ID_A)$

$\Longrightarrow$ the calculations is the following:

$(CR_A)^3 = \left[[H(n_A, ID_A)]^{d_{CA}}\right]^3 \bmod n_{CA} = [H(n_A, ID_A)]^{d_{CA}\cdot 3} \bmod n_{CA}.$

Since $d_{CA} = 3^{-1} \bmod \varphi(n_{CA})$, $d_{CA} \cdot 3 = 1$

$\Longrightarrow (CR_A)^3 = [H(n_A, ID_A)]^1 \bmod n_{CA} = H(n_A, ID_A)$

$[H(n_A, ID_A) < n_{CA} = p_1 * p_2]$.

5.2 Implementation Results

This section presents implementation results pertaining to all three methods described in stage B, in which the approached node proves its validity, providing a comparison in terms of timing and energy resources.

The methodology descried in stage A and all of the three methodologies discussed in stage B were implemented on the Intel Mote 2 [6] platform. The latter employs the Intel PXA271 XScale Processor running at a clock frequency ranging from 13 MHz to 416 MHz. The core frequency can be dynamically set in software, allowing the designer to carefully adjust the timing/power trade-off so as to optimize performance of a particular application. Functions were taken from the TinyECC package [49]. The latter targeted the MICA2 platform and provided a basic library of ECC-based functions, including scalar multiplication and exponentiation operations. Customizations for the XScale processor, including 32-bit operation optimizations, were carried out. In addition, supplementary functions, such as efficient Montgomery arithmetic, were added. All codes are written in NesC running on the TinyOS operating system. Nodes exchange

messages using a 2.4 GHz embedded low-power radio transceiver. In all of the implementations depicted below, the clock frequency was 312MHz, scalars (for key transport usage) were 1024 bits, scalars (for ECC based computations) were 160 bits, and points on the curve (for ECC based computations) were 160 bits for each of the vertices. It should be noted that 160-bit keys in ECC are equivalent, from a cryptocomplexity perspective, to 1024-bit keys in RSA.

Self-certified fixed-key generation, excluding DoS mitigation, takes one dynamic point-by-scalar multiplications. Hence, on the Intel mote 2 platform the process takes 42 msec to complete and consumes 22 mJ at each node (see table 3.3)

Stage A, in which Alice proves her validity to Bob, is identical regardless of the methodology chosen in Part B. For the latter, it is imperative to understand the overhead involved in calculating $t^{d_A} \bmod n_A = m$. All other calculations and communications are relatively negligible. For a key size of 1024 bits (for both d_A and n_A) the computation took Alice 230 msec and drained 105.8 mJ. On the other hand, Bob's calculation of $(CR_A)^3 \bmod n_{CA}$ took 1.02 msec and drained only 0.469 mJ. The energy consumed when Alice performs her procedure is three orders of a magnitude larger than the energy consumed when Bob performs his. The results were 230 msec and 105.8 mJ, substantiating the effectiveness of the procedure proposed. All other computations and transmissions are relatively negligible.

In stage B, when using key transport, Bob is required to perform the exact symmetrical procedure that Alice preformed in stage A, i.e., $S_B = y^{d_B} \bmod n$. Hence Bob will spend 230 msec and 105.8 mJ. In the validation process, Alice will perform the following two calculations: $(CR_B)^e \bmod n_{CA}$, $(S_B)^3 \bmod n_B$ and will have spent 2.04 msec and 0.938 mJ. When using ECDSA, the important computations are point-by-scalar multiplications. As described above, Bob preforms one point-by-scalar multiplication while Alice performs two. When using the self-certified fixed-key method, each of the nodes performs one point-by-scalar multiplication. Each multiplication takes about 42 msec and consumes 22 mJ. Please see tables 5.1 and 5.2 for details. All other computations and transmissions are relatively negligible.

As expected, using key transport as means of certification is not beneficial in resource constrained environments. In other applications, where resources are not scarce, key transport can be extremely useful, since there is no need for additional elliptic curve calculations (as in ECDSA and fixed-key scenarios). It should be noted that calculations of the key transport

Table 5.1: Time (msec) and energy (mJ) consumed while performing stage A and stage B for 1024-bit RSA and 160-bit ECC on the Intel mote 2 patform for 312 MHz core clock

	Time (msec)		**Energy (mJ)**		Total	
	Alice	Bob	Alice	Bob	Time	Energy
Stage A	230	1.02	105.8	0.469	231.02	106.27
Stage B						
Key Transport	2.04	230	0.938	105.8	232.04	106.738
ECDSA	83	42	44	22	125	66
Fixed Key	42	42	22	22	84	44

Table 5.2: Total time (msec) and energy (mJ) consumed by each of the three techniques for ephemeral key establishment in the DoS mitigation

	Time (msec)	**Energy (mJ)**
Total consumption	Both stages	Both stages
Key Transport	463.06	213.01
ECDSA	356.02	172.27
Fixed Key	315.02	150.27

method in both stages are almost symmetric when it comes to the computational load that Bob and Alice have. When comparing ECDSA to fixed-key generation, we come to the conclusion that the fixed-key method is more efficient since it implies a 15% energy gain and reduced time.

Chapter 6

Light-weight Arithmetic Algorithms

Given that wireless sensor nodes are very limited in energy, memory and processing resources, it is imperative to efficiently utilize the existing resources in any computational task performed. The cryptography field employs numerous fundamental arithmetic algorithms. Two such algorithms, the Montgomery arithmetic for modular multiplication and the generation of a modular multiplicative inverse, are treated in this chapter. As indicated in previous chapters, the core calculation (for example, when establishing a secret key) using ECC is the point-by-scalar multiplication. As part of this elaborate calculation, there is a need to compute modular multiplication between two scalars. It is this modular scalar-by-scalar multiplication that utilizes the above mentioned algorithms. In order to multiply we need to use the Montgomery arithmetic for modular multiplication, and as part of this arithmetic, the modular multiplicative inverse of a scalar is computed. While simulating various key generation schemes, it became apparent that these two algorithms are frequently called upon, which raised the clear need for their efficient lightweight realization. The following sections provide a description of the algorithms and their proposed light-weight implementations in detail.

6.1 A System-Level Efficient Utilization of the Montgomery Procedure

Modular exponentiation constitutes a fundamental building block in public key cryptographic operations. In Elliptic Curve Cryptography (ECC) implementations, the exponentiation is

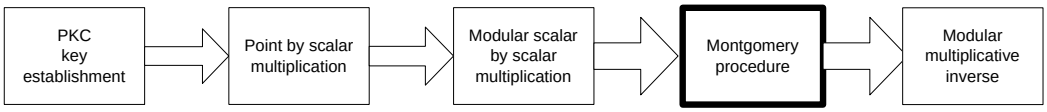

Figure 6-1: The chain of importance in the PKC key establishment process, illustrating the essential role of the Montgomery procedure

associated with a scalar curve-point multiplication, which we shall refer to as the *ECC exponentiation core* [44], [55]. Such cores facilitate the execution of several modular multiplication operations needed in various applications, ranging from digital signatures to key establishments. The challenge faced by the designers concern the efficient execution of these modular multiplication operations when providing for a specific application. A typical chain of importance is depicted in figure 6-1.

This section pertains to public key cryptographic certification. In *explicit certification*, the CA signs the association between a user's ID/attributes (and other administrative details), and the user's public key. The CA does so by using standard digital signature techniques in which the generated signature acts as a certificate attesting to the validity of the values submitted by users. In *implicit certification* [32],[63],[7], authenticity of parameters submitted by a user is established as an inherent part of executing the application served by these parameters, where the CA's public key is one of the inputs to the process.

Next, a description of the steps to be carried out by the CA when generating explicit or implicit certificates is provided. These steps are carried out in order to simplify the execution of the operations subsequently performed by the verifiers of the issued certificates. In view of the above, we consider the modular multiplication operations that employ an exponentiation core, treating the core as a given module.

An exemplifying case of explicit certification (that is, a direct digital signature generated by the CA) is the Elliptic Curve Digital Signature Algorithm [4] (ECDSA). An exemplifying case for implicit certification is the self-certification methodology presented in [7]. The framework presented here can be applied to a wide range of public key cryptographic applications.

6.1.1 Observations on Montgomery Arithmetic Constructs

We next briefly review the Montgomery modular multiplication of two scalars, p and q, to yield $p \cdot q \ mod \ n$ [48]. The procedure (also presented in 5.1.3) processes values represented in base 2^m, that is, they consist of m-bit coefficients. Let v_i denote the i^{th} coefficient of an integer v, where the least significant coefficient is denoted by v_0. We let k be the number of m-bit coefficients in p and q. Moreover, we let r denote the value of $-n_0^{-1} \bmod 2^m$ (To clarify, n_0 represents the least significant coefficient of the modulus n). The Montgomery procedure can thus be described as follows:

Procedure 1: *Montgomery modular multiplication*

begin

$s = 0, t = 0, v = 0$

for $i = 0$ to $k - 1$

$t = s + q_i \cdot p$

$u = (t_0 \cdot r)_0$

$v = t + u \cdot n$

$s = \frac{v}{2^m}$

end (for loop)

if $s > n$ then $s = s - n$

end

In the above, $s = p \cdot q \cdot 2^{-mk} \bmod n$. However, our purpose is to obtain the result: $p \cdot q \bmod n$. The operation $t = s + q_i \cdot p$ that is executed k times, where k is the number of m *bit* coefficients comprising the multiplier q, is the pure multiplication operation $p \cdot q$. Similarly, the operation $v = t + u \cdot n$ is also a pure multiplication. The only difference between multiplication of the form $t = s + q_i \cdot p$ and that of the form $v = t + u \cdot n$ is the fact that the q_i*'s* are known *a priori*, while the u's are generated dynamically. There are two additional m-bit operations in the procedure, involving single coefficients, both intended to guarantee that t and v do not exceed $k + 1$ coefficients (with an exception of a possible 1-bit overflow). These m *bit* operations effectively render the entire procedure a 'character-level shift-and-add operation'. Intuitively, the division essence of modular multiplication is realized at the m *bit* character-level, by the dynamic generation of the u's. This holds since division is based on dynamic decisions

concerning the subtraction of a divisor, while in multiplication, the decisions of whether to add the multiplicand or not are known in advance and depend on the particular structure of the multiplier.

As indicated before, in Procedure 1, $s = p \cdot q \cdot 2^{-mk} \bmod n$. However, our purpose is to obtain the result: $p \cdot q \bmod n$. In order to eliminate the undesired multiplicative factor 2^{-mk}, we apply Procedure 1 for a second time, where p and q are replaced by $p \cdot q \cdot 2^{-mk} \bmod n$ (the result obtained from the first round) and 2^{2mk} - a system constant that can be precalculated and stored. The output of the second stage will be the desired result of $p \cdot q \bmod n$.

As indicated, Procedure 1 does not yield the final desired result of $p \cdot q \bmod n$, but rather the value $p \cdot q \cdot 2^{-mk} \bmod n$, necessitating the execution of Procedure 1 for a second time. This introduces a major deviation from the ultimate goal of calculating $p \cdot q \bmod n$ using *two pure m bit multiplications*, hence using Procedure 1 only once. Here we will show how this ultimate goal can be achieved for the case of executing modular multiplication operations that employ an exponentiation core within the framework of public key cryptographic certification. In particular, we provide detailed treatment of both explicit and implicit certification. The former is shown for digital signature generation and verification based on ECDSA, while the latter pertains to self-certified Diffie-Hellman key-establishment [7].

A fundamental observation is that if the CA employs Procedure 1 *only once* during certificate generation, that is, it executes two pure *m bit* multiplications, then the certificate verifiers are also required to employ Procedure 1 only once. In these cases, the undesired multiplicative factor of the Montgomery arithmetic operation has no effect, thus the procedure is executed only once.

6.1.2 Explicit Certification based on ECDSA

The ECDSA [4] algorithm enables signature generation and signature verification procedures. As a preliminary stage, all participants agree on specific curve parameters and a generating point, $\mathbf{G}$, of order $ord\mathbf{G}$ (exponents are calculated modulo $ordG$). The signer's private key is s while his public key is the curve point $\mathbf{W} = s \times \mathbf{G}$, where $\times$ represents a scalar curve-point multiplication. Letting f denote the message to be signed, the following outlines the signature generation and verification procedures:

Procedure 2: *ECDSA signature generation*

1. The CA generates a random number u and calculates $\mathbf{V} = u \times \mathbf{G}$

2. Let c be a scalar representation of $\mathbf{V}$, utilizing the standard procedure specified in [4],

the CA calculate $d = u^{-1} \cdot (f + s \cdot c)\ mod\ ord\mathbf{G}$ to obtain the pair (c, d) as the signature.

Procedure 3: *ECDSA signature verification*

1. Compute $h = d^{-1}\ mod\ ord\mathbf{G}$, $m = f \cdot h\ mod\ ord\mathbf{G}$ and $q = c \cdot h\ mod\ ord\mathbf{G}$, based on the verifier's knowledge of $\mathbf{G}$, $ord\mathbf{G}$, $\mathbf{W}$, f, c, and d.

2. Obtain the curve point $\mathbf{P} = m \times \mathbf{G} + q \times \mathbf{W}$.

   Let c' be the scalar representation of $\mathbf{P}$, using the standard procedure specified in [4].

3. If c'$= c$ then the signature is determined to be valid.

In both procedures (yielding the signature generation and signature verification operations) there is a need to perform a modular multiplication, hence necessitating the need to execute procedure 1 twice.

We next describe how the execution of the control operations can be carried out using pure multiplications in the framework of explicit certification.

Observing the expression $d = u^{-1} \cdot (f + s \cdot c)\ mod\ ord\mathbf{G}$ in Procedure 2, where the pair (c, d) is the generated signature, we let the signing CA calculate $h = d^{-1} mod ord\mathbf{G} = u \cdot (f + s \cdot c)^{-1} mod ord\mathbf{G}$. Furthermore, we let $t = c \cdot 2^{-mk} \bmod ord\mathbf{G}$, which can be calculated by multiplying c by 1, using Procedure 1. Thus, the explicit certificate is the pair (t, h). Subsequently, the verifier performs the following:

1. Computes

$$m = f \cdot h \cdot 2^{-mk} mod ord\mathbf{G} \text{ and } q = c \cdot h \cdot 2^{-mk} mod ord\mathbf{G}. \tag{6.1}$$

2. Computes the curve point $\mathbf{P} = m \times \mathbf{G} + q \times \mathbf{W}$

3. Let t' be the scalar representation of $\mathbf{P}$, if $t\prime = t$ then the signature is determined to be valid.

As can be observed from the above, the calculation of the modular multiplication represented in equation 6.1 is achieved by executing Procedure 1 only once. No operation was made to remove the multiplicative factor 2^{-mk}, as this is compensated by using the new certificate (t, h).

6.1.3 Implicit Certification in Self-certified Procedures

Let d be the CA's private key and $\mathbf{R}$ its public key (where $\mathbf{R} = d \times \mathbf{G}$). User i, served by the CA, is denoted by N_i, with a *private* key, x_i. h_i is a random scalar, unique to each user, generated by the CA for the purpose of calculating the private keys. $\mathbf{U}_i$ is defined as the *public* key of user i, and $H(v, \mathbf{W})$ is a scalar obtained by performing a hash transformation on the scalar v and curve-point $\mathbf{W}$. To issue $N_i's$ public and private keys, the CA generates the random scalar h_i. Respectively, the two keys are given by

$$\begin{aligned} \mathbf{U}_i &= h_i \times \mathbf{G} \\ x_i &= [H(ID_i, \mathbf{U}_i) \cdot h_i + d] \ mod \ ord\mathbf{G}. \end{aligned} \tag{6.2}$$

$N_j's$ public and private keys are issued in a similar manner.

Fixed Key-Generation

We briefly review the steps to be executed in achieving self-certified fixed-key generation (as discussed in section 2.5.2).

1. N_i and N_j exchange the pairs $(ID_i, \mathbf{U}_i)$ and $(ID_j, \mathbf{U}_j)$, respectively

2. K_{ij} (generated by N_i) $= x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}]$, while K_{ji} (generated by N_j) $= x_j \times [H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}]$. Note that *all* scalar by scalar multiplications are calculated modulo $ord\mathbf{G}$.

   All modular multiplications needed for the above procedures are performed *mod* $ord\mathbf{G}$. Since $\mathbf{U}_i = h_i \times \mathbf{G}$, the argument $H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j$ necessitates the calculation of $H(ID_j, \mathbf{U}_j) \cdot h_i \times \mathbf{G}$, where $H(ID_j, \mathbf{U}_j) \cdot h_i$ is a modular multiplication necessitating

the execution of Procedure 1 twice.

These operations can be the result of executing Procedure 1 *only once*, based on the following:

Key issuing by the CA: $N_i's$ public and private keys are

$$\begin{aligned} \mathbf{U}_i &= h_i \times \mathbf{G} \\ x_i &= \Big[H(ID_i, \mathbf{U}_i) \cdot h_i \cdot 2^{-mk} + d\Big] \, mod \; ord\mathbf{G} \end{aligned} \tag{6.3}$$

The argument $H(ID_i, \mathbf{U}_i) \cdot h_i \cdot 2^{-mk}$ is derived by performing Procedure 1 *only once*. Similar calculations hold for user N_j.

Key-generation: In order to generate the fixed-key, the two steps are:

1. N_i and N_j exchange the pairs $(ID_i, \mathbf{U}_i)$ and $(ID_j, \mathbf{U}_j)$, respectively
2. $K_{ij} = x_i \times [H(ID_j, \mathbf{U}_j) \cdot 2^{-mk} \times \mathbf{U}_j + \mathbf{R}]$, and $K_{ji} = x_j \times [H(ID_i, \mathbf{U}_i) \cdot 2^{-mk} \times \mathbf{U}_i + \mathbf{R}]$.

Hence $K_{ij} = x_i \times [H(ID_j, \mathbf{U}_j) \cdot 2^{-mk} \times \mathbf{U}_j + \mathbf{R}] = x_i \times (H(ID_j, \mathbf{U}_j) \cdot 2^{-mk} \cdot h_i + d) \times \mathbf{G}$ $= x_i \cdot x_j \times \mathbf{G}$. Similar considerations apply for K_{ji}. A key confirmation should now follow. If indeed $K_{ij} = K_{ji}$, then we observe fixed-key self-certification.

It should be noted that all modular multiplications performed by N_i and N_j were based on executing Procedure 1 *only once*. No operation was made to remove the multiplicative factor 2^{-mk}, as this is compensated by the CA carrying out the same operation during the key issuing process.

Ephemeral Key-Generation

As described in section 2.5.3, in order to achieve a self-certified ephemeral key-generation, users N_i and N_j perform the following:

1. N_i and N_j generate a random scalar pv_i and pv_j, respectively
2. N_i calculates the ephemeral value $\mathbf{EV}_i = pv_i \times \mathbf{G}$. N_j calculates the ephemeral value $\mathbf{EV}_j = pv_j \times \mathbf{G}$

3. N_i and N_j exchange the values $(ID_i, \mathbf{U}_i, \mathbf{EV}_i)$ and $(ID_j, \mathbf{U}_j, \mathbf{EV}_j)$, respectively

4. K_{ij} (generated by N_i)$= pv_i \times [H(ID_j, \mathbf{U}_j)] \times \mathbf{U}_j + \mathbf{R}] + (x_i + pv_i) \times \mathbf{EV}_j$,

   K_{ji} (generated by N_j) $= pv_j \times [H(ID_i, \mathbf{U}_i)] \times \mathbf{U}_i + \mathbf{R}] + (x_j + pv_j) \times \mathbf{EV}_i$

   All modular multiplications in the above procedures are preformed *mod ord*$\mathbf{G}$. Again, since $\mathbf{U}_i = h_i \times \mathbf{G}$, the argument $H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j$ necessitates the calculation of $H(ID_j, \mathbf{U}_j) \cdot h_i$ which is a modular multiplication necessitating the execution of Procedure 1 twice. .

These operations can be the result of executing Procedure 1 *only once*, based on the following:

Key issuing by the CA: Considering users N_i and N_j,we have

$$\begin{aligned} \mathbf{U}_i &= h_i \times \mathbf{G} \\ x_i &= \left[H(ID_i, \mathbf{U}_i) \cdot h_i \cdot 2^{-mk} + d\right] \ mod\ ord\mathbf{G}, \\ \mathbf{U}_j &= h_j \times \mathbf{G} \\ x_j &= \left[H(ID_j, \mathbf{U}_j) \times h_j \times 2^{-mk} + d\right] \ mod\ ord\mathbf{G}. \end{aligned} \tag{6.4}$$

Once again, the distinction in the proposed improvement lies in the arguments $H(ID_j, \mathbf{U}_j) \cdot h_j \cdot 2^{-mk}$ and $H(ID_j, \mathbf{U}_j) \cdot h_j \cdot 2^{-mk}$ derived by performing Procedure 1 *only once*. The following steps are followed in order to generate the self-certified ephemeral key:

1. N_i and N_j generate a random scalar pv_i and pv_j, respectively

2. N_i calculates the ephemeral value $\mathbf{EV}_i = pv_i \times \mathbf{G}$. N_j calculates the ephemeral value $\mathbf{EV}_j = pv_j \times \mathbf{G}$

3. N_i and N_j exchange the values $(ID_i, \mathbf{U}_i, \mathbf{EV}_i)$ and $(ID_j, \mathbf{U}_j, \mathbf{EV}_j)$, respectively

4. K_{ij} (generated by N_i) $= pv_i \times [H(ID_j, \mathbf{U}_j)] \cdot 2^{-mk} \times \mathbf{U}_j + \mathbf{R}] + (x_i + pv_i) \times \mathbf{EV}_j$,

5. K_{ji} (generated by N_j) $= pv_j \times [H(ID_i, \mathbf{U}_i)] \cdot 2^{-mk} \times \mathbf{U}_i + \mathbf{R}] + (x_j + pv_j) \times \mathbf{EV}_i$

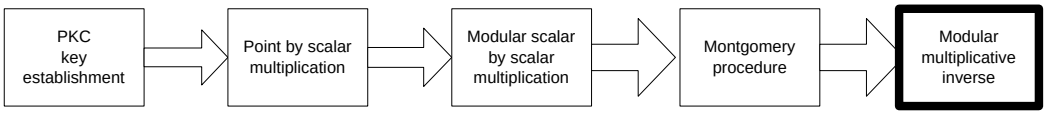

Figure 6-2: The chain of importance in the PKC key establishment process, illustrating the important role of the modular multiplication procedure

Hence $K_{ij} = pv_j \times [H(ID_j, \mathbf{U}_j) \cdot 2^{-mk} \times \mathbf{U}_j + \mathbf{R}] + (x_j + pv_j) \times \mathbf{EV}_i = pv_j \times [H(ID_j, \mathbf{U}_j) \cdot 2^{-mk} \cdot h_i + d] \times \mathbf{G} + (x_j + pv_j) \times \mathbf{EV}_i = pv_i \times x_j \times \mathbf{G} + x_i \times pv_j \times \mathbf{G} + pv_i \times pv_j \times \mathbf{G}$. Similar considerations apply for K_{ji}.

A key confirmation should now follow. If indeed $K_{ij} = K_{ji}$,then we observe an ephemeral key self-certification.

As in the case of fixed-key generation, all modular multiplications performed by N_i and N_j were based on executing Procedure 1 *only once*. No operation was applied to remove the multiplicative factor 2^{-mk}, as this is compensated by the CA performing the same operation during key issuing.

Procedure 1 ends with the need to subtract the modulus n from the resultant value s, in cases of $s > n$. While this may be computationally negligible, it somewhat disrupts the goal of treating Procedure 1 as a pure two-multiplications operation. In this respect, it is observed that for the cases in which the value generated by the procedure is intended to be used as a scalar in an ECC scalar-point multiplication (or as an exponent in public key cryptographic implementations over integers), this final operation in Procedure 1 can be discarded. In other words, the value s can be kept as it is, even if it exceeds s. This is permitted since the multiplication of s by a point reduces it modulo $ordG$.

6.2 Modular Multiplicative Inverse

When attempting to calculate a modular multiplication between two scalars, we utilized the Montgomery algorithm as indicated in the previous section. In order to perform these computations, one must calculate a negative modulo 2^m multiplicative inverse. The multiplicative inverse modulo 2^m of an odd value b, denoted by $b^{-1} \bmod 2^m$, is a value r where $b \cdot r = 1 \bmod 2^m$.

Without loss of generality it can be assumed that $b < 2^m$, (i.e., b is an m-bit value), since otherwise b is first taken $\bmod 2^m$ (i.e., only the least m significant bits in the binary representation of b are selected) before proceeding with the calculation of r. The binary representation of r also consists of m bits. Consequently, this section is divided into two parts. The first pertains to the calculation of a modular 2^m multiplicative inverse of a parameter, i.e. $b^{-1} \bmod (2^m)$, and the second addresses the calculation of a negative modular 2^m multiplicative inverse of a parameter, i.e. $-b^{-1} \bmod (2^m)$.

All of these basic computations are extremely useful and desired tools in the world of number theory, in general, and in the field of cryptography in particular. Classic example of scenarios in which the modulus is a power of 2 are the Montgomery modular multiplication [48] and the *exact division problem* [64], [40]. In the latter, one has a list of odd word-sized numbers each divisible by k, and there is a need to divide all of them by k. Here it is needed to compute $b^{-1} \bmod 2^m$ where m is the number of bits in a word.

This section presents procedures for calculating a multiplicative inverse modulo 2^m using a novel mathematical approach that is not an extension or modification of known approaches such as: [24] , [46], [37], [61], [47]. These known algorithms that perform the above calculations involve a number of steps linearly proportional to the number of characters in a number. This chapter demonstrates an efficient methodology for a character-based computation of modular multiplicative inverses that involve a power of 2, reducing these calculations to a logarithmic number of steps. Hence, the amount of calculations involved and the computational resources are tremendously decreased form an $O(m)$ complexity to a $O(logm)$ complexity, where m is the amount of characters in the number.

6.2.1 Calculating the expression $b^{-1} \bmod (2^m)$

This section presents a procedure for calculating the multiplicative inverses modulo 2^m based on a novel mathematical approach. The procedure is suitable for software implementation on a general-purpose processor. When counting the total number of word-level processor multiplications, the computational effort involved in calculating a multiplicative inverse is two thirds that of a single multiplication of m-bit values, in addition to a few word-level multiplications. For standard processor word sizes, the number of these additional multiplications does not exceed

12.

As indicated above, the multiplicative inverse modulo 2^m of a value b, is a value r, where $b \cdot r = 1 \bmod 2^m$. By definition, b is not necessarily restricted to be an $m-bit$ integer, although in practice this is usually the case. Hence, the desired result is $r = b^{-1} \bmod 2^m$. Cases where the modulus is a power of 2 are encountered in various modular multiplication procedures (such as in the Montgomery algorithm for modular multiplications). These calculations are straightforward but at the same time consume a great deal of computation.

Straightforward Calculations

The following are several known or straightforward algorithms for calculating $b^{-1} \bmod 2^m$. The first is taken from Dusse and Kaliski [24].

Algorithm 1 *Dusse and Kaliski's method for calculating* $r = b^{-1} \bmod 2^m$

$y_1 = 1$

for $i = 2$ *to* m

if $b \cdot y_{i-1} < 2^{i-1} \bmod 2^i$

then $y_i = y_{i-1}$

else $y_i = y_{i-1} + 2^{i-1}$

The value of y_m *is the desired result* $r = b^{-1} \bmod 2^m$.

The following is a straightforward method for obtaining the same result.

Algorithm 2 *Calculating* $r = b^{-1} \bmod 2^m$ *by controlled additions of left-shifts of* b

Since $b \cdot r = 1 \bmod 2^m$, *the* m *least significant bits of the complete product* $b \cdot r$ *are of the form* 000...01. *To obtain* r, *one should therefore add selected left-shifts of the binary representation of* b *such that the addition of the shifts generates the* m *least significant bits* 000...01. *The coefficients of the added left-shifts form the binary representation of* r. *(We form here a standard shift-and-add multiplication, where the multiplicand* b *is known, and the* m *least significant bits of the product are given. From this we recover the multiplier* r, *bit by bit, starting from the LSB.)*

This algorithm is always feasible to implement since b is odd, having 1 as the least significant bit. Hence, by adding selected left-shifts of b to an accumulated sum that starts with b, one can

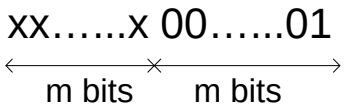

Figure 6-3: Illustration of the identity expression $r \cdot b = x \cdot 2^m + 1$

always generate a value whose m least significant bits are of any form, including 000...01. In other words, Algorithm 2 is executed by 'sliding' b left across an accumulated sum, such that the least significant bit of b generates the bits of the given product, one at a time, from right to left.

As an example, let k_H and k_L respectively denote the higher half and lower half of the binary representation of an $2m - bit$ value, k. Each of these halves is an $m - bit$ value. The relation $b \cdot r = 1 \bmod (2^m)$ means that $b \cdot r = x \cdot 2^m + 1$ for some integer x. This means that the following $m - bit$ identity hold: $(b \cdot r)_L = (000...01)_2$. Figure 6-3 visually illustrates this identity.

Since the multiplication $r \cdot b$ has the structure indicated above, calculating r becomes rather straightforward.

1. Lets look at the following example where we want to calculate the modular 2^4 multiplicative inverse of 3, for $m = 4$. In other words, we would like to calculate r where $r = 3^{-1} mod \left(2^4\right)$. We begin by shifting and adding the binary representation of 3, i.e. $(0011)_2$, until the least significant part of the additions will consist of the sequence "0001". Figure 6-4 shows the manner by which additions are employed. Since the coefficients of the added shifts are 2^0, 2^1 and 2^3, the representation of r is: $2^0 + 2^1 + 2^3 = (11)_{10} = (1011)_2 \Rightarrow 11$, which is the modular 2^4 multiplicative inverse of 3.

The above bit-wise straightforward procedure for calculating a multiplicative inverse modulo 2^m is rather time consuming. If such a calculation is seldom performed in a given application, it may be justified to utilize the bit-wise procedure. However, when there is a need to frequently perform such a calculation, it is advisable to try and execute this procedure by performing character-level multiplication, fully exploiting the processing power of a given processor.

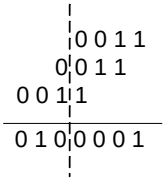

Figure 6-4: Shifting and adding the binary representation of 3 until the least significant part of the additions includes four bits in the format "0001"

Calculating $b^{-1} \bmod 2^m$ can be based on initially computing $2^{-m} \bmod b$ and then recovering $b^{-1} \bmod 2^m$ by one division of a $2m$-bit value by an m-bit value as shown by Algorithm 3a followed by Algorithm 3b below.

Algorithm 3 *Calculating* $r = b^{-1} \bmod 2^m$ *in two steps*

Algorithm 3a: Calculating $2^{-m} \bmod b$ *by* m *successive divisions of 2* $\bmod b$

$d = 1$

for $i = 1$ *to* m

if d *is odd then* $d = d + b$

$d = d/2$

The final value of d *is, of course,* $2^{-m} \bmod b$.

Algorithm 3b: Recovering $r = b^{-1} \bmod 2^m$ *out of* $s = 2^{-m} \bmod b$

The desired $r = b^{-1} \bmod 2^m$ *is recovered out of* $s = 2^{-m} \bmod b$ *as follows:*

$t = s \cdot 2^m$

$u = (t - 1)/b$

$r = 2^m - u$

To realize why Algorithm 3b is valid note that the relation $s = 2^{-m} \bmod b$ means that $t = s \cdot 2^m = u \cdot b + 1$ and the value u, calculated in the second step of Algorithm 3b, is therefore an integer. That is, $u \cdot b = s \cdot 2^m - 1$. Taking both sides of the latter relation $\bmod 2^m$ yields the congruence $u \cdot b = -1 \bmod 2^m$. That is, $(-u) \cdot b = 1 \bmod 2^m$, and therefore $-u = r = b^{-1} \bmod 2^m$. However, $-u = (2^m - u) \bmod 2^m$, which completes the validity proof for Algorithm 3b.

Another method of calculating modular multiplicative inverses is *The Extended Euclid Algorithm* [46]. It is important to note that this algorithm treats odd moduli. Therefore, when

computing $r = b^{-1} \bmod 2^m$, we first calculate the multiplicative inverse of 2^m modulo the odd b, and then exchange the role of the two values 2^m and b. Algorithm 3a is precisely the Extended Euclid Algorithm when used in calculating $2^{-m} \bmod b$, while Algorithm 3b is the procedure of exchanging 2^m and b. Methods of calculating modular multiplicative inverses further include the *Montgomery Inverse Algorithm* [37], [61]. This algorithm, which treats an odd moduli, consists of two phases of which the second is identical to Algorithm 3a. Since the modulus is odd, calculating $r = b^{-1} \bmod 2^m$ further necessitates the procedure of Algorithm 3b. Other methods presented in the literature for calculating $b^{-1} \bmod 2^m$ include lookup table techniques such as that proposed in [47], which are inherently non-algorithmic.

Efficient Calculations

Let k_H and k_L denote the higher half and lower half of the binary representation of a $2i$-bit value k, respectively. Each of these halves is an i-bit value. For example, in the relation $b \cdot r = 1 \bmod 2^i$, i.e., $b \cdot r = x \cdot 2^i + 1$ for some integer x, $(b \cdot r)_L = (000...01)_2$. In order to calculate $r = b^{-1} \bmod 2^m$, we first consider the calculation of $p = q^{-1} \bmod 2^{2i}$ given $q_L^{-1} \bmod 2^i$.

Theorem 1 *Given b and r as i-bit values, where $r = b^{-1} mod\ 2^i$, and given q as a $2i$-bit value where $q_L = b$, the value $p = q^{-1} \bmod 2^{2i}$ can be efficiently obtained by calculating its lower half, p_L, and its higher half, p_H (both are i-bit values independently).*

The Lower Half of p can be calculated as

$$p_L = r, \tag{6.5}$$

while the Higher Half of p can be calculated as

$$p_H = -\left[\left[(r \cdot b)_H + (r \cdot q_H)_L\right] \cdot r\right] \bmod 2^i, \tag{6.6}$$

where p is formed by the concatenation between p_H and p_L such that $p = p_H | p_L$.

Proof. The relation $p = q^{-1} \bmod 2^{2i}$ suggests that $p \cdot q = 1 \bmod 2^{2i}$. From the multiplication $p \cdot q$, we derive that

$$(p_H p_L) \cdot (q_H q_L) = p_H \cdot q_H \cdot 2^{2i} + (p_H \cdot q_L + p_L \cdot q_H) \cdot 2^i + p_L \cdot q_L, \tag{6.7}$$

as depicted in figure 6-5. Therefore, the $2i$ least significant bits of $p \cdot q = (p_H p_L) \cdot (q_H q_L)$ can be written as

$$\left[(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^i + p_L \cdot q_L\right] \bmod 2^{2i}. \tag{6.8}$$

Since $p \cdot q = 1 \bmod 2^{2i}$,

$$\left[(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^i + p_L \cdot q_L\right] = 1 \bmod 2^{2i}. \tag{6.9}$$

Calculating the Lower Part of p

Since the i least significant bits of $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^i + p_L \cdot q_L$ are also the i least significant bits of $p_L \cdot q_L$, only $p_L \cdot q_L$ dictates that the i least significant bits of $(p_H \cdot q_L + p_L \cdot q_H)_L \cdot 2^i + p_L \cdot q_L$ are 1 (see figure 6-5 for details). That is,

$$p_L \cdot q_L = 1 \Longrightarrow p_L \cdot q_L = 1 \bmod 2^i \Longrightarrow p_L = q_L^{-1} \bmod 2^i. \tag{6.10}$$

Since $q_L = b$ and $r = b^{-1} \bmod 2^i$,

$$p_L = r.$$

Calculating the Higher Part of p

Given that the $2i$ least significant bits of the expression $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^i + p_L \cdot q_L$ form the value 1,

$$[(p_H \cdot q_L + p_L \cdot q_H)_L + (p_L \cdot q_L)_H] \bmod 2^i = 0 \tag{6.11}$$

(see the middle section in Figure 6-5 for details). That is,

$$(p_H \cdot q_L)_L = -(p_L \cdot q_L)_H - (p_L \cdot q_H)_L \bmod 2^i. \tag{6.12}$$

Since the least significant component, k_L, of a number k is also $k \bmod 2^i$,

$$p_H = -[[(p_L \cdot q_L)_H + (p_L \cdot q_H)_L] \cdot q_L^{-1}] \bmod 2^i. \tag{6.13}$$

As $q_L = b$, $r = b^{-1} \bmod 2^i$ and $p_L = r$, we conclude that

$$p_H = -[[(r \cdot b)_H + (r \cdot q_H)_L] \cdot r] \bmod 2^i. \tag{6.14}$$

■

Example 1 *To illustrate this result, we refer to the following example. For simplicity, all numbers are displayed in hexadecimal base. Our purpose is to find* $p = q^{-1} \bmod 2^{32}$*, given that* $q = 99F8A5EF$ *and* $q_L = b$*, where* $r = b^{-1} = (A5EF)^{-1} = 290F$ *mod* 2^{16}*. Using the efficient calculation method described in Theorem 1, we can derive that* $p_L = r = 290F$*. In order to calculate* p_H *we are required to follow three easy steps:*

1. $(r \cdot b)_H = (290F \cdot A5EF)_H = 1A9D$
2. $(r \cdot q_H)_L = (290F \cdot 99F8)_L = BD88$
3. $p_H = -[((r \cdot b)_H + (q_H \cdot r)_L) \cdot r] \bmod 2^{16} = -((1A9D + BD88) \cdot 290F) \bmod 2^{16} = 68D5$

In order to obtain the final result, p, all that is left is the concatenation. Hence, $p = p_H | p_L = 68D5290F$*. The result can be checked by validating the following equality* $p \cdot q = 68D5290F \cdot 99F8A5EF = 3F0D37FD00000001 = 1$ *mod* 2^{32}.

Theorem 1 specifies the three operations: $(r \cdot b)_H$, $(r \cdot q_H)_L$ and $[(r \cdot b)_H + (r \cdot q_H)_L] \cdot r \bmod 2^i$. All values are i-bit long. The first operation involves the multiplication of i-bit values. On the other hand, when performing the latter two, only the lower half of the generated product is

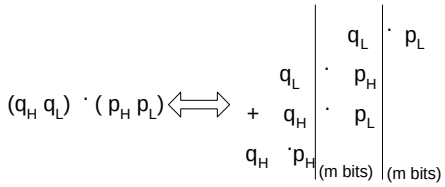

Figure 6-5: The format of $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L$

needed, requiring half of the full multiplication effort. This leads us to the following conclusion:

Conclusion 1 *The execution of Theorem 1 altogether involves the computation of two multiplications of i-bit values.*

Theorem 1 is used in reaching the final goal of calculating $b^{-1} \bmod 2^m$. The fundamental process of the theorem can be iteratively repeated $\log m$ times, by doubling the number of bits when calculating the modular inverse in each iteration. At the end of such process, we are left with $b^{-1} \bmod 2^m$. An illustrative example of this process is discussed in the subsection below.

Computational Efforts

We begin with an introductory graphical illustration, depicted in Figure 6-6, demonstrating the proposed procedure for calculating $b^{-1} \bmod 2^m$, where m = 8-bits. Finding the multiplicative inverse of the number $''abcdefgh''$ comprises of the following steps:

0. Finding the multiplicative inverse of $'h'$ modulo 2^1. Note that $'h'$ is 1 since the number $''abcdefgh''$ is odd. For the same reason, the inverse of $'h'$ is also 1;

1. Finding the multiplicative inverse of $''gh''$ modulo 2^2, by exploiting the result from step (0);

2. Finding the multiplicative inverse of $''efgh''$ modulo 2^4 using the result obtain in step (1) and, finally,

3. Finding the multiplicative inverse of the entire number $''abcdefgh''$ modulo 2^8 by using the result from step (2).

It is noted that steps (1)-(3) are achieved by using Theorem 1.

Overall Computational Efforts in Terms of Word-Level Processor Multiplications

In the case treated by this section, $r = b^{-1} \bmod 2^m$ is calculated, where m is a power of 2. Consider the case where each digit in the illustration of Figure 6-6 is a processor-word of k-bits. In practice, k is a power of 2. Letting $n = \frac{m}{k}$, the representation of r and b consists of n words.

The process illustrated in Figure 6-6 suggests that $\log n$ consecutive doublings of $h^{-1} \bmod 2^k$ yield $r = (abcdefgh)^{-1} \bmod 2^m$ (where h is the right most word of $abcdefgh$). In the j-th stage,

$j = 0, 1, ..., \log n - 1$, a value consisting of 2^j words is doubled in size. Based on the observation in Conclusion1, such doubling requires two multiplications of values consisting of 2^j words. One such multiplication requires $(2^j)^2 = 2^{2j}$ single-word multiplications. The entire process thus involves $2 \cdot \sum_{j=0}^{\log n - 1} 2^{2j}$ single-word multiplications. It should be noted that $2 \cdot 2^{2 \cdot (\log\ n-1)} = 2 \cdot (\frac{n}{2})^2$, i.e., during the last stage, a value consisting of $\frac{n}{2}$ words is doubled, requiring $2 \cdot (\frac{n}{2})^2$ single-word multiplications. The total number of single-word multiplications required when calculating $r = b^{-1} \bmod 2^m$, given the initial value $h^{-1} \bmod 2^k$, is therefore $2 \cdot [\sum_{j=0}^{\log n - 1} 2^{2j}] = \frac{2(n^2-1)}{3}$, where n^2 is the number of single-word multiplications executed when multiplying two m-bit operands. By definition, each such operand consists of n words.

Let us now evaluate the computational effort involved in calculating $h^{-1} \bmod 2^k$. This calculation in itself requires $\log k$ executions of the process in Theorem 1, doubling 1-bit, 2-bit, ..., $\frac{k}{2}$-bit, values. Since this section also concerns software implementations using a general-purpose processor, these $\log k$ small values should each be considered as a complete processor word. Thus, when implementing Conclusion 1, this effort requires $2 \cdot \log k$ single-word multiplications.

As a result, when calculating $r = b^{-1} \bmod 2^m$, the overall number of single-word multiplications is:

$$\frac{2(n^2 - 1)}{3} + 2 \log k \tag{6.15}$$

Standard sizes for a processor word are $8, 16, 32, 64$-bits, for which $\log k = 3$, 4, 5, 6, where n^2 is the number of single-word multiplications executed when multiplying two m-bit operands. It is concluded that the computational effort involved in calculating the multiplicative inverse of an odd value b modulo 2^m is two thirds of one multiplication of m-bit values, plus a negligible number of single-word multiplications which in practical cases does not exceed 12 ($= 2 \log k$).

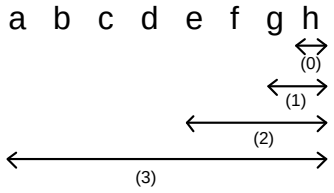

Figure 6-6: Finding the multiplicative inverse of the number represented by "abcdefgh".

Performance Comparison

The performance of the presented procedure for calculating $b^{-1} \bmod 2^m$ is compared to that of other procedures based on two criteria: (1)the ability for software execution on a general-purpose processor, and (2)the computational effort, measured in terms of the overall number of executed word-level multiplications.

The presented procedure was shown to be suitable for software implementation on a general-purpose processor. When counting the total number of word-level multiplications, it was shown that the computational effort involved in calculating $b^{-1} \bmod 2^m$ is two thirds of one multiplication of m-bit values, plus a few word-level multiplications, whose number in practice does not exceed 12.

Algorithm 1 [24] is executed in $m-1$ steps, each one involving the multiplication operation $b \cdot y_{i-1}$, where the multiplicand consists of m bits. The procedure can be naturally performed in software using a general-purpose processor. In contrast, this section presents an approach in which the overall number of single-word multiplications is equivalent to a single m-bit multiplication, also executed on a general-purpose processor.

Algorithm 2 is executed in m steps, involving shifts-and-adds. Altogether this is equivalent to one multiplication of m-bit values. Shifts and decisions are being made at the bit-level. Therefore, on the one hand, the computational effort is equivalent to one m-bit multiplication, like the procedure presented in this section. On the other hand, the bit-level considerations in Algorithm 2 pose implementation difficulties on a general-purpose processor, a disadvantage which the procedure presented in this section overcomes.

Algorithm 3 is executed in m steps involving shifts-and-adds. Shifts and decisions in Algorithm 3a are made at the bit-level. Therefore, the procedure presented in this section outperforms Algorithm 3 as far as both discussed criteria are concerned.

It should be noted that the Montgomery modular multiplication procedure [48] yields the value $x \cdot y \cdot 2^{-m} \bmod b$, for m-bit operands. Algorithm 3a can therefore be replaced by Montgomery procedure, for $x = y = 1$ (yielding $2^{-m} \bmod b$). The replacement of Algorithm 3a by a Montgomery multiplication can possibly have the advantage of running in software on a general processor, utilizing word-level multiplications. Regardless of this effort, there is still a need to perform a division by an m-bit value, as required by Algorithm 3b. Therefore, the procedure

presented in this section significantly outperforms this case too.

We next compare the procedure proposed in this section to the *Extended Euclid Algorithm [46]* and to the *Montgomery Inverse Algorithm [37], [61].* As shown in this section, Algorithm 3a is precisely the Extended Euclid Algorithm when used in calculating $2^{-m} \mod b$, while Algorithm 3b is the procedure of exchanging the role of 2^m and b, yielding the result $b^{-1} \bmod 2^m$. Hence, Algorithms 3a+3b are the implementation of the Extended Euclid Algorithm for the case treated in this section. The Montgomery Inverse Algorithm treats an odd modulus. Therefore, an implementation of this algorithm in calculating $b^{-1} \bmod 2^m$ would mean that $2^{-m} \bmod b$ is calculated first, and $b^{-1} \bmod 2^m$ is then recovered by executing Algorithm 3b. Accordingly, Algorithm 3b is still needed when using the Montgomery Inverse Algorithm. Furthermore, Algorithm 3a is the second phase of the Montgomery Inverse Algorithm. Therefore, when calculating $b^{-1} \bmod 2^m$, the execution of Algorithms 3a+3b is more efficient than the Montgomery Inverse. It is concluded that the procedure presented in this section for calculating $b^{-1} \bmod 2^m$, which outperforms Algorithms 3a+3b as shown above, outperforms the Extended Euclid Algorithm and the Montgomery Inverse Algorithm.

Beside the practical advantages (shown above) of the presented procedure, it should be noted again that the mathematical approach taken by this section is different from those treated in all of the above referenced papers.

Treating the Case where m is not a Power of 2

Let $a^{-1} = b \mod p$ and q be a divisor of p. Basic observations in number theory show that $[a \bmod q]^{-1} = b \mod q$. For clarification, letting $a = 11$, $p = 14$ and $q = 7$, it is evident that $11^{-1} = 9 \bmod 14$, and $4^{-1} = 2 \bmod 7$.

The procedure presented in section 6.2.1 treats the case where m is a power of 2 (allowing for the described consecutive doublings). In cases where m is not a power of 2, the binary representation of b consists of $n = 2^{\lceil \log\ m \rceil}$ bits, and the value $w = b^{-1} \bmod 2^n$ is calculated in $\lceil \log\ m \rceil$ steps, based on the procedure of section 6.2.1. Consequently, $r = w \bmod 2^m$ is the required result $b^{-1} \bmod 2^m$. An important observation pertains to the fact that calculating $w \bmod 2^m$ can be done by taking the m least significant bits in the binary representation of w.

As a concluding remark, one can see that various procedures have been proposed in the past for calculating $b^{-1} \bmod 2^m$. Some lend themselves to software execution on a general processor utilizing word-level multiplications. The overall computational effort associated with such procedures is equivalent to more than one multiplication of m-bit operands. Other procedures, having overall computational effort equivalent to one multiplication of m-bit operands, are executed on a bit-level and are not suitable for software implementation using a general-purpose processor (i.e., they cannot efficiently utilize word-level multiplications). This subsection presented a procedure for calculating $r = b^{-1} \bmod 2^m$ which on the one hand is suitable for software implementation using a general-purpose processor, while on the other hand it utilizes processor-word multiplications. When counting the total number of such multiplications, the computational effort involved in calculating r is two thirds of one multiplication of m-bit values, plus a few word-level multiplications, whose number in practice does not exceed twelve. This combines the best individual performances of known procedures, when considering minimal computational effort and ability for software execution on a general-purpose processor. To the best of the authors' knowledge, the mathematical principle on which the proposed procedure relies is novel and is not an extension or modification of known approaches.

6.2.2 Calculating the Expression $-b^{-1} \bmod (2^m)$

This specific calculation of $-b^{-1} \bmod (2^m)$ occurs frequently when performing any general modular calculations, in particular when establishing point-by-scalar multiplications using ECC.

The *negative* modular 2^m multiplicative inverse of a parameter, b, is defined as $-b^{-1} \bmod (2^m)$. Accordingly, if we define the parameter r' to be the negative modular multiplicative inverse of b, then $r' = -b^{-1} \bmod (2^m)$. Hence, $b \cdot r' \bmod (2^m) = -1$, or $b \cdot r' = x \cdot 2^m - 1$ (where x is an arbitrary number). From the latter expression we understand that the product $r' \cdot b$ is a multiplicative of 2^m, -1. To that end, this product is of the form of XF in hexadecimal format, where X is an undefined character of m bits and $F = (111....1)_2$ with the length of m bits as well.

If, theoretically, we would have had the *positive* modular 2^m multiplicative inverse of b, i.e. r, (efficient calculation of which is showed in the section above), then the best and simplest way of calculating the negative value would be to simply use the following identity: $r' = 2^m - r$.

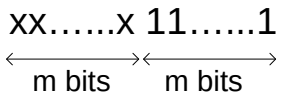

Figure 6-7: The format of the expression $r' \cdot b = x \cdot 2^m - 1$

Unfortunately, having prior knowledge of r is almost never the case, hence efficient calculations are required here as well.

Straightforward Calculations

Let k_H and k_L respectively denote the higher half and lower half of the binary representation of a $2m - bit$ value k. Each of these halves is an $m - bit$ value. The relation $b \cdot r' = 1 \bmod (2^m)$ means that $b \cdot r' = x \cdot 2^m - 1$ for some integer x. This implies that the m-bit expression $(b \cdot r')_L = (111...111)_2$, as illustrated in figure 6-7.

Since the multiplication $r' \cdot b$ has the structure indicated above, calculating r is rather straightforward:

1. Add shifts of the binary representation b such that the addition of the shifts generates the m least significant bits 111...1

2. *Adding the coefficients of the added shifts form the binary representation of* r'

Lets look at the following example in which we want to calculate the negative modular Multiplicative Inverse of 13 given that $m = 4$. In other words, we would like to calculate r' where $r' = -13^{-1} mod \left(2^4\right)$. We will begin by shifting and adding the binary representation of 13, i.e. $(1101)_2$, until the least significant part of the additions will include four 1's (see figure 6-8).

Since the coefficients of the added shifts are 2^0, 2^1 and 2^3, the representation of r' is $2^0 + 2^1 + 2^3 = (11)_{10} = (1011)_2 \Rightarrow 11$, which is the negative modular multiplicative inverse of 13. As indicated above, this bit-wise straightforward procedure for calculating a negative multiplicative inverse modulo 2^m is rather time consuming. Hence, a better and more efficient way of calculations is highly desirable.

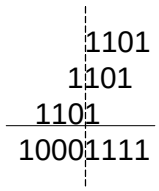

Figure 6-8: Shifting and adding the binary representation of the number 13 until the least significant part of the additions includes four 1's

Efficient Calculations

Let b and r' be given $m\ -bit$ values, where $r' = -b^{-1} \bmod 2^m$. Let q be a given $2m - bit$ value where $q_L = b$. It is next shown how to efficiently calculate $p = -q^{-1} \bmod 2^{2m}$, utilizing the above. We note the following known values:

1. The value of r', where $r' = -b^{-1} \bmod 2^m$
2. The value of q
3. The fact that $q_L = b$

   As in section 6.2.1, obtaining p is partitioned into three steps; calculating the lower half, i.e. p_L (m-bits), calculating of the higher half, i.e. p_H (m-bits) and finalizing the entire value p.

   - The lower half of p is given by

     $$p_L = r$$

   - The higher half is given by

     $$p_H = [[(r' \cdot b)_H + [r' \cdot q_H]_L + 1] \cdot r'] \bmod 2^m$$

   - Calculate p by performing the following:

     $$p = p_H | p_L$$

Proof of these derivations will proceed the following example.

As an example, let's consider the case of finding $p = -q^{-1} \bmod 2^{32}$, given that $q = 99F8A5EF$ and $q_L = b$, where $r' = D6F1 = -A5EF^{-1} mod 2^{16}$. Using the efficient calculation described above, we can derive that $p_L = r = D6F1$. In order to calculate p_H we need to follow three easy steps:

1. $(r' \cdot b)_H = (D6F1xA5EF)_H = 8B51$
2. $(r' \cdot q_H)_L = (99F8xD6F1)_L = 4278$
3. $p_H = [[(r' \cdot b)_H + [r' \cdot (q)_H]_L + 1] \cdot r'] \bmod 2^{16} = (CDCAxD6F1) \bmod 2^{16} = 972A$

In order to receive an accurate result of p, all that is left is the concatenation, hence $p = p_H|p_L = 972A|D6F1 = 972AD6F1$. The result can be checked using the following identity: $p \cdot q = 972AD6F1 \cdot 99F8A5EF = 5AEB6DF1FFFFFFFF = -1 mod 2^{32}$. As in the previous case, this method can be used to efficiently and easily calculate the negative inverse modulo 2^m of a large value, by first calculating the negative modular multiplicative inverse of a much smaller part of this given value. The first calculation (the given r' in the above expressions) can be used for further calculations. Evaluation of the larger number will be based upon the known r' and will increase by powers of 2.

Proof

All of the parameters in the following proof represent binary numbers. By knowing r', b and m, where $r' = -b^{-1} \bmod 2^m$, we can show that calculating the expression $p = -q^{-1} \bmod 2^{2m}$ where $q_L = b$, can be done by using the efficient calculation described above. The identity $p = -q^{-1} \bmod 2^{2m}$ implies that $p \cdot q = -1 \bmod 2^{2m}$. Since $p \equiv p_H p_L$ and $q \equiv q_H q_L$, the expression $p \cdot q$ takes the form $(p_H p_L) \cdot (q_H q_L) = p_H \cdot q_H \cdot 2^{2m} + (p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L$ (as in the previous case). Requiring that $(p_H p_L) \cdot (q_H q_L) = -1 \bmod 2^{2m}$ means that $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L = -1 \bmod 2^{2m}$. That is, the least significant $2m$ bits of $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L$ should all be $1\prime s$.

Calculating the Lower Part of p We note the fact that the least significant m bits of $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L$ are the least significant m bits of $p_L \cdot q_L$. Hence, only $p_L \cdot q_L$ dictates that the m least significant bits of $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L$ will be

1's. The last observation suggests that $p_L \cdot q_L = -1 \bmod 2^m$. Since $q_L = b$ we conclude that $p_L = -b^{-1} \bmod 2^m$. Since it is given that $r' = -b^{-1} \bmod 2^m$ it follows that $p_L = r'$.

Calculating the Higher Part of p According to the preceding result, the expression $(p_H \cdot q_L + p_L \cdot q_H) \cdot 2^m + p_L \cdot q_L$ has m least significant $1\imath s$, for $p_L = r'$. The second portion of m (to the left of these m least significant 1's) are identified as $[p_H \cdot q_L + p_L \cdot q_H + (p_L \cdot q_L)_H] \bmod 2^m$. Given that $[p_H \cdot q_L + p_L \cdot q_H + (p_L \cdot q_L)_H] = -1 \bmod 2^m$, the bits identified as $[p_H \cdot q_L + p_L \cdot q_H + (p_L \cdot q_L)_H] \bmod 2^m$ should also be 1. Since $p_L = r'$, we receive the following expression: $[p_H \cdot q_L + r' \cdot q_H + (r' \cdot q_L)_H] = -1 \bmod 2^m$. Furthermore, from the identity $r' = -b^{-1} \bmod 2^m$ it follows that $b = -(r')^{-1} \bmod 2^m$. Substituting this into the preceding relation yields: $-p_H \cdot (r')^{-1} \bmod 2^m = -r' \cdot q_H - (r' \cdot b)_H - 1 \bmod 2^m$, that is: $p_H = r' \cdot [r' \cdot q_H + (r' \cdot b)_H + 1] \bmod 2^m$.

To avoid handling large values, the expression $r' \cdot (q)_H$ can be reduced mod 2^m inside the brackets, yielding the final relation: $p_H = r' \cdot [(r' \cdot q_H)_L + (r' \cdot b)_H + 1] \bmod 2^m$.

Chapter 7

Summary of Contributions

This dissertation has described novel foundations needed to develop a more resilient infrastructure for securing resource-constrained sensor networks. In particular, practical algorithms for accelerating the computations involved in both pairwise and group key establishments have been of primary focus. Basic cryptographic arithmetic operations have been revised in order to accommodate the unique attributes of WSNs. This has yielded interesting results that have potential impact to the field of applied cryptography as a whole. The following summarizes the primary contributions made in this dissertation.

7.1 Self-Certified Public Key Generation with Off-loading Provisioning

A primary contribution made in this work pertains to the introduction of off-loading techniques in the context of ECC-based self-certified public key generation. Off-loading non-secure tasks from a component having low resources to an assisting node was described in detail, applied to both fixed and ephemeral key generations. Off-loading allows for prolonging of the network lifetime, by distributing the computational effort across more nodes. This directly answers the challenge of minimizing resources, while adapting to the ad-hoc topology of the network. Successful implementations of these algorithms on both the Intel mote 2 platform and TelosB mote accentuate the practical aspects of this contribution. The positive impact of off-loading with regard to network lifetime and overall reduction in computation time was evaluated through

simulations that addressed randomly deployed sensor nodes. To complement the pairwise key generation scheme, an extension to group key generation has been proposed, introducing a linear increase in energy and time with respect to the network density.

7.2 Delay-efficient Group Key Generation

A more efficient group key generation scheme has been proposed for WSN applications where the overall process time is critical. The scheme is an extension to the Burmester and Desmedt (BD) algorithm, with a trade off between energy consumption and overall key generation latency when compared to the group key generation process established when using the self-certified pairwise key method. Executing an independent signature generation/verification procedure performed when validating ephemeral values is not necessary in this group key generation. As in the case of pairwise key generation, identification of malicious elements in the network is achieved via key confirmation processes.

7.3 Resource-efficient Denial-of-service Mitigation

In order to address the fundamental issue of attacks that target the energy of sensor nodes by repeatedly requesting key establishments, an efficient scheme for mitigating such attacks has been proposed and analyzed. The key idea is to shift the computational burden on the node initiating the session, rather than that which is being approached. A careful study of various alternatives for achieving this goal has been carried out.

7.4 Light-weight Arithmetic Algorithms

In order to retain high-performance when resources are scarce, it is imperative to revisit the implementations of fundamental cryptographic functions. Here, a substantial contribution was made in implementing Montgomery operations more efficiently by considering the network involved. Improvements of the modular multiplicative inverse calculation were also introduced. For the latter, the computational resources were dramatically decreased from a complexity of $O(m)$ to that of $O(\log m)$. We have proven that this procedure for calculating the modular

multiplicative inverse is suitable for software implementation using a general-purpose processor, while utilizing processor-word multiplications. This achievement was made without incurring substantial memory cost (as is typical in other time/space trade-off techniques).

Relevant Publications

The following is a list of publications covering the contributions made in this dissertation:

1. **O. Arazi**, H. Qi, "*An Efficient Computation of Inverse Multiplicative Operations*," *in revision for publication in IEEE Transactions on Computer*s.

2. **O. Arazi**, H. Qi, D. Rose, "*A Public Key Cryptographic Method for Denial of Service Mitigation in Wireless Sensor Networks*," 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), June, 2007, San Diego, CA.

3. **O. Arazi**, H. Qi, "*Toward Mitigating Denial of Service Attacks in Power-Constrained Sensor Networks*," 2007 *Cyber Security and Information Infrastructure Research Workshop*, Oak Ridge National Lab (ORNL), TN, May 14-15, 2007.

4. **O. Arazi**, D. Rose, H. Qi, B. Arazi, "*Self-Certified Public Key Generation on the Intel Mote 2 Sensor Network Platform*" *3rd Annual IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, Sept., 2006, Reston, VA.

5. **O. Arazi**, H. Qi, "*Load-Balanced Key Establishment Methodologies in Wireless Sensor Networks*," *International Journal of Sensor Networks (IJSN)* (Special issue on Security for Sensor Networks), Vol. 1, No. 2, April 2006.

6. **O. Arazi** and H. Qi, "*Self-Certified Group Key Generation for Ad Hoc Clusters in Wireless Sensor Networks*," in Proc. of the *14th IEEE International Conference on Computer Communications and Networks (ICCCN)*, San Diego, CA, Oct. 17-19, 2005.

7. **O. Arazi**, B. Arazi, H. Qi, I. Elhanany, D. Rose, "*Self-Certified Public Key Cryptography for Resource-Constrained Sensor Networks*," *2006 CyberSecurity and Information Infrastructure Research Workshop, Oak Ridge National Lab (ORNL)*, TN, May 10-11, 2006.

8. B. Arazi, I. Elhanany, **O. Arazi**, and H. Qi, "*Revisiting Public Key Cryptography for Wireless Sensor Network*," *IEEE Computer Magazine*, pp. 85-87, Nov. 2005.

Bibliography

Bibliography

[1] "Certicom ECC challenge," *Certicom, Inc.*, available at URL: http://www.certicom.com/index.php.

[2] "Mica2 datasheet," *Crossbow Technology, Inc.*, available at URL: www.xbow.com.

[3] "Telosb datasheet," *Crossbow Technology, Inc.*, available at URL: www.xbow.com.

[4] "IEEE std 1363-2000: Specifications for public key cryptography," Tech. Rep., 2000.

[5] E. T. A. Shamir, "Factoring large numbers with the twirl device," in *Crypto 2003, LNCS 2729*, 2003, pp. 1–26, springer-Verlag.

[6] R. Adler, M. Flanigan, J. Huang, R. Kling, N. K. L. Nachman, C.-Y. Wan, and M. Yarvis, "Intel mote 2: an advanced platform for demanding sensor network applications," in *SenSys 2005: Proceedings of the 3rd international conference on Embedded networked sensor systems*, 2005, pp. 292–298.

[7] B. Arazi, "Certification of dl/ec keys," in *Proc. of the IEEE P1363 Study Group for Future Public-Key Cryptography Standards*, May 1999.

[8] O. Arazi, I. Elhanany, D. Rose, and H. Q. B. Arazi, "Self-certified public key generation on the intel mote 2 sensor network platform," in *Third Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, SECON 06*, 2006.

[9] H. B. B. Chen, K. Jamieson and R. Morris, "Span: An energy-efficient co-ordination algorithm for topology maintenance in ad hoc wireless networks," in *MobiCom*, Rome, Italy, July 2001, pp. 70–84.

[10] D. Boneh, N. Modadugu, and M. Kim, *Generating RSA keys on a handheld using an untrusted server*. New York: Springer-Verlag, 2000.

[11] G. Borriello, "Ten emerging technologies that will change the world," *Technology Review*, February 2003.

[12] E. Bresson, O. Chevassut, D. Pointcheval, and j. Quisquater, "Provably authenticated group diffie-hellman key exchange," in *ACM Conference on Computer and Communications Security*, 2001, pp. 255–264.

[13] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *EUROCRYPT '94, LNCS 950*, 1995, pp. 275–286.

[14] ——, "A secure and scalable group key exchange system," *Information Processing Letters*, vol. 94, pp. 137–142, 2005.

[15] A. H. S. K. U. V. C. Blundo, A. D. Santis and M. Yung, "Perfectly-secure key distribution for dynamic conferences," in *Advances in Cryptology - CRYPTOŠ92*. Springer-Verlag, Berlin, 1992, p. 471Ű486.

[16] L. A. M. C. Blundo and D. R. Stinson, "radeoffs between communication and storage in unconditionally secure schemes for broadcast encryption and interactive key distribution," in *Advances in Cryptology - CRYPTOŠ96*. Berlin: Springer-Verlag, Berlin, 1996, p. 387Ű400.

[17] Certicom, "Online elliptic curve cryptography tutorial," *Certicom, Inc.*, available at URL: /www.certicom.com.

[18] H. Chan and A. Perrig, *PIKE: Peer intermediaries for key establishment in sensor networks*, 2005.

[19] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, Washington DC, USA, 2003, pp. 197–214.

[20] P. K. D. Carman and B. Matt, "Constraints and approaches for distributed sensor network security," in *NAI Labs*.

[21] P. N. D. Liu, "Establishing pairwise keys in distributed sensor networks," in *10th Computer and Communications Security*, 2003.

[22] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.

[23] W. Du, J. Deng, Y. S. Han, and P. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, Washington DC, USA, October 2003, pp. 42–51.

[24] S. R. Dusse and B. S. Kaliski, "A cryptographic library for the motorola dsp5600," in *Advances in Cryptology–EUROCRYPT'90*, 1990, pp. 230–244, lNCS - Springer-Verlag.

[25] R. Dutta, R. Barua, and P. Sarkar, "Provably secure authenticated tree based group key agreement," in *Proceedings of the 6th International Conference on Information and Communications Security, LNCS 3269*, 2004, pp. 92–104.

[26] W. B. W. P. E. Barker, W. Barker and M. Smid, "Recommendation for key management - part 1: General national institute of standards and technology," *NIST Special Publication*, August 2005.

[27] Energizer, "AA battery datasheet," Tech. Rep., 2005. [Online]. Available: available at: http://data.energizer.com/PDFs/E91.pdf

[28] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*, Washington, DC, November 2002, pp. 41–47.

[29] A. M. Eskicioglu and E. J. Delp, "A key transport protocol based on secret sharing applications to information security," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 4, pp. 816–824, Novemeber 2002.

[30] D. Estrin, "Embedded everywhere: A research agenda for networked systems of embedded computers," *National Research CouncilReport*, 2001.

[31] A. Fiat and M. Naor, "Broadcast encryption," in *Advances in Cryptology - CRYPTOŠ93*. Springer-Verlag, Berlin, 1993, p. 480Ű491.

[32] M. Girault, "Self-certified public keys," in *Advances in Cryptology–EUROCRYPT'91*, March 1991, pp. 491–497, lNCS - Springer-Verlag.

[33] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient routing protocols for wireless microsensor networks," in *Proc. 33rd Hawaii International Conference on System Sciences (HICSS)*, January 2000, pp. 3005–3014.

[34] F. Information, "Announcing the standard for digital signature standard (dss)," *Processing Standards Publication 186*, May 1994.

[35] J. Jonsson and B. Kaliski, "Public-key cryptography standards (pKCS) #1: RSA cryptography specifications version 2.1," United States, 2003.

[36] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *ASIACRYPT '96, LNCS 1163*, 1996, pp. 36–49.

[37] B. S. Kaliski, "The montgomery inverse and its applications," *IEEE Transactions On Computers*, vol. 44, no. 8, pp. 1064–1065, August 1995.

[38] Y. Kim, , A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, vol. 7, pp. 60–96, 2004.

[39] Y. Kim, A. Perrig, and G. Tsudik, "Group key agreement efficient in communication," *Communications of the ACM*, vol. 53, no. 7, pp. 905–921, July 2004.

[40] N. Koblitz, *A Course in Number Theory and Cryptography (Graduate Texts in Mathematics)*. Springer, September 1994.

[41] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Designs, Codes and Cryptography*, vol. 19, pp. 173–193, 2000.

[42] J. F. Kurose and K. W. Rose, *Computer Networking: A top down approach featuring the Internet*. Addison-Wesley, 2005.

[43] P. Lee, J. Lui, and D. Yau, "Distributed collaborative key agreement and authentication protocols for dynamic peer groups," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 263–276, 2006.

[44] D. Malan, "ECCM," available at URL: www.eecs.harvard.edu / malan/eccm.shtml.

[45] D. Malan, M. Welsh, and M. D. Smith, "A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography," in *Proc. of 1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, Santa Clara, CA, October 2004.

[46] J. L. Massey, "Cryptography: Fundamentals and applications," *Advanced Technology Seminars*, February 1993.

[47] D. Matula, A. Fit-Florea, and M. Thornton, "Table lookup structures for multiplicative inverses modulo 2/sup k," in *ARITH-17, 17th IEEE Symposium on Computer Arithmetic*, June 2005, pp. 156 – 163.

[48] P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[49] P. Ning and A. Liu, "TinyECC: Elliptic curve cryptography for sensor networks," Tech. Rep., 2005, http://discovery.csc.ncsu.edu/software/TinyECC.

[50] O.Arazi and H. Qi, "Load-balanced key establishment methodologies in wireless sensor networks," *International Journal of Sensor Networks, IJSN*, vol. 1, no. 2, April 2006.

[51] N. I. of Standard and Technology, "Federal information data encryption standard," *Processing Standard Publications 46-2*, 1993, available at URL: http://www.itl.nist.gov/fipspubs/fip46-2.htm.

[52] ——, "Data encryption standard fact sheet," 1999, available at URL: http://csrc.nist.gov/cryptval/des.txt.

[53] ——, "Draft federal information processing standard (fips) 46-3 data encryption standard (des) and requests for comments," 1999, available at URL: http://csrc.nist.gov/cryptval/des/fr990115.htm.

[54] ——, "Advanced encryption standard (AES)," *Federal Information Processing Standard 197*, November 2001, available at URL: http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[55] N. Peng, "Tinyecc: Elliptic curve cryptography for sensor networks (version 0.1)," available at URL: http://discovery.csc.ncsu.edu/software/TinyECC/index.html.

[56] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, June 2004.

[57] H. Qi and Y. Xu, "Decentralized reactive clustering for collaborative processing in sensor networks," in *Proc. of the IEEE 10th International Conference on Parallel and Distributed Systems (ICPADS)*, vol. 91, no. 8, Newport Beach, CA, July 2004, pp. 54–61.

[58] H. Qi, Y. Xu, and X. Wang, "Mobile-agent-based collaborative signal and information processing in sensor networks," in *Proceedings of the IEEE*, vol. 91, no. 8, August 2003, pp. 1172–1183.

[59] D. W. R. Molva, G. Tsudik, *Security and Privacy in Ad-hoc and Sensor Networks*, ser. Lecture Notes in Computer Science, 2005, vol. 3813.

[60] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[61] E. Savas and C. K. Koc, "The montgomery modular inverse - revised," *IEEE Transactions On Computers*, vol. 49, no. 7, pp. 763–766, July 2000.

[62] A. S. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall, 2003.

[63] S. Tsujii and T. Itoh, "An id-based cryptosystem based on the discrete logarithm problem," *IEEE J. on Selected Areas in Communications*, vol. 7, pp. 467–473, 1989.

[64] J. Tudor, "An algorithm for exact division," *Journal of Symbolic Computation archive*, vol. 15, no. 2, pp. 169 – 180, February 1993.

[65] S. Vanstone, A. J. Menezes, and M. Qu, "Key agreement and transport protocol with implicit signatures," *United States Patent 5,896,455*, 1999.

[66] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "Tinypk: Securing sensor networks with public key technology," in *Proceedings of the Second ACM Workshop on Security of Ad Hoc and Sensor Networks*, Washington DC, USA, 2004, pp. 59–64.

Vita

Ortal Arazi was born in Pretoria, South Africa, on June 11, 1973. She attended Ben-Gurion University in Israel, where she received a Bachelor of Science and Master of Science degrees in Electrical and Computer Engineering, as well as a Master in Business Administration. In 2003, she has tought undergraduate computer engineering courses at San Jose State University, California. In 2004, she began doctorate studies in Electrical and Computer Engineering at the University of Tennessee, Knoxville, where she received the Doctor of Philosophy degree in November 2007.