A Public Key Cryptographic Method for Denial of Service Mitigation in Wireless Sensor Networks

O. Arazi[1,2], H. Qi[1], D. Rose[1]

[1]Department of Electrical and Computer Engineering
University of Tennessee
Knoxville, TN 37996-2100

[2]Cyberspace Sciences & Information Intelligence Research Group (CSIIR)
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6418

***Abstract*— The challenging characteristics of sensor nodes, including the constrained resources, the ad-hoc nature of their deployment and the vulnerability of wireless media, pose a need for unique security solutions. The advantages of Public Key Cryptography (PKC) for sensor network security are widely acknowledged and include resilience, scalability and decentralized management. Recent work has indicated that PKC is feasible in the wireless sensor network (WSN) environment, paving the way for many new security services and opportunities. 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. In this paper, we propose a novel RSA-based framework for combating DoS attacks in WSN by ensuring that the malicious party will exhaust its resources prior to exhausting those of its counterparts. Under the proposed approach, the mathematical operations performed by the malicious party require two or three orders of magnitude more resources than those required by the attacked party. We also present three methodologies for establishing an ephemeral key, 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.**

I. Introduction

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 [1], [2], [3], [4]. 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 every 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 reside, 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 communication 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 [5], [6], [7], [8], hosting collaborative processing, has been a solution in achieving both fault tolerance and scalability. 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.

A fundamental requisite for security, other than providing data confidentiality and authentication, is Denial of Service (DoS) prevention. The computational effort involved in per-

forming PKC calculations is 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 DoS attacks, when compared to traditional security methods that require less resources. In particular, if a malicious party attacks a sensor node by futile repetitive requests to establish a joint secret key, the resources of the attacked node will be exhausted quite rapidly. Combating DoS attacks is the last frontier to be conquered prior to making PKC deployment standard security practice in sensor networks.

This paper focuses on a public key cryptographic approach for mitigating DoS attacks in sensor networks. In particular, the computational asymmetry in RSA signature generation schemes is exploited to yield a resource-efficient authentication mechanism which helps overcome DoS attacks. The rest of the paper is structured as follows. Section II provides an overview of public key cryptographic methodologies introduced for sensor network environments. In Section III, a description of Elliptic Curve Cryptography-based key generation and authentication schemes are provided. Section IV focuses on the proposed framework for DoS mitigation, which is embedded within the key establishment process. In Section V we provide implementation results, while in section VI the conclusions are drawn.

II. Public-Key Cryptography for Sensor Networks

A simple solution for key establishment has been proposed in the literature, in which a single network-wide shared key is used. However, the capture of a single node could easily reveal the network secret key. A current mainstream effort consists of random key pre-distribution [4], [9], [10], [11], [12], [13], in which a different set of pre-established keys is issued to each node, thereby reducing the probability that capturing one node will jeopardize the entire network. A trivial key pre-distribution scheme is to allow each node to hold $N-1$ secret pairwise keys, each of which is known only to the node and to one of the other $N-1$ nodes (assuming there are N nodes in the network). However, the constrained memory resources and the difficulty in adding new nodes to the network limit the effectiveness of this general scheme.

Other researchers have extended the original notion of key pre-distribution to include a statistical element. In particular, methods such as those proposed in [14] assume that each sensor node receives a random subset of keys drawn from a large key pool. To agree on a key for communication, two nodes find one common key within their subsets and use that key as their shared secret key. Additional information, such as data concerning the position and/or geographical distribution of the sensor nodes, can be used to further improve the key pre-distribution concept [9]. However, these schemes only offer a partial solution with respect to scalability, cryptographic robustness and the ability to append and revoke security attributes. For example, scalability can be 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).

The problems identified in key pre-distribution schemes triggered an in-depth study of public key cryptographic key-establishment for sensor networks. Correspondingly, there has been a growing effort in promoting public-key cryptography in sensor networks [15], [16], [17], [18], [19], [20]. Elliptic Curve Cryptography (ECC) [21] emerges as a suitable public key cryptographic foundation for sensor networks, providing high security for relatively small key sizes. Malan *et al.* [15] demonstrated an implementation of point by scalar multiplication over elliptic curves, which is the basic and most time-consuming operation in ECC, on MICA2 motes [22]. Ning *et al.* [23] further improved the implementation performance by at least a factor of 5 using well-know optimization techniques. These recent results indicate that the execution of ECC operations in sensor nodes is feasible, with predictable improved performance.

A major theme in public key cryptographic applications concerns *certification*, which ensures the safe exchange of public keys. A Certification Authority (CA) issues a certificate, attesting to the connection between a user's public key and his ID. Verifying a certificate requires an explicit reference to the CA's public key. An authentication procedure, which is based on certification, therefore needs the following values: the user's public key, his ID, the certificate, and the CA's public key. The latter is considered to be universal and known to all parties, while the first three values are unique to each user.

To further improve the computational efficiency of the key establishment procedure, *self-certified* public key cryptography was proposed [24], in which a user submits its ID along with its public key, but does not submit an explicit certificate, thereby reducing communication and management overheads. In identity-based systems [25], the user's public key is its actual ID, which avoids the need for any public value other than the user's ID. Nevertheless, an explicit reference to the CA's public key is still required. In the context of key establishment, self certification means that the authentication of values submitted by the participating parties is *inherently embedded within the process of generating the session key*. This is in contrast to the case of explicit certification, whereby authenticity of the submitted values has to be verified prior to the actual generation of the joint session key. A well known self-certified key generation method is the MQV [26], adopted by the NSA [27].

III. ECC-based Key generation and Authentication

A. Notation and Formulation

We begin by reviewing the foundations for ECC-based key generation and authentication, as introduced by the authors in [28]. Our mathematical foundations rely on ECC cryptographic techniques pertaining to operations over a finite group of points in which the discrete log problem applies. In order to describe the formalism for efficient two-node Diffie Hellman (DH) key generation, we must first define some notations and terminologies. A group-point is hereby denoted by a capital letter in bold font and a scalar will be presented in regular lowercase letters. Multiplication of a point by a scalar (e.g., $s \times \mathbf{P}$) will be referred to as an exponentiation, where s is the exponent. The intractability of a discrete log operation means that given the points $\mathbf{P}$ and $s \times \mathbf{P}$, the complexity of finding s is exponential.

The following notations are employed throughout the remainder of this paper: $\mathbf{G}$ - 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.

B. Keys Issued to Nodes by the CA

The private and public keys discussed here are issued by the CA to all nodes in the network. We will begin our discussion by focusing only on keys issued to N_i. As indicated above, the CA holds a pair of keys (private (d) and public ($\mathbf{R}$)). By using d, ID_i, h_i, a hash function and $\mathbf{G}$, the CA establishes the pair of private and public keys issued to node i. We consider two scenarios for issuing the private key (x_i), and the public key ($\mathbf{U}_i$) of node i. The node's private key x_i, used in the following applications, can be derived by either scenarios described in this section. In the first scenario, the CA knows the node's secret keys. In this case, N_i's private key (x_i), and the public value ($\mathbf{U}_i$) can be generated as follows. First, the CA generates a random scalar h_i and calculates $h_i \times \mathbf{G}$. Next, the CA then generates node i's public and private keys by performing:

$$\begin{aligned} \mathbf{U}_i &= h_i \times \mathbf{G} \\ x_i &= [H(ID_i, \mathbf{U}_i) \times h_i + d] \ mod\ ord\mathbf{G} \end{aligned} \quad (1)$$

The CA issues the values x_i and $\mathbf{U}_i$ to N_i, at which time N_i can establish the validity of the values issued to him by checking whether $x_i \times \mathbf{G} = H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}$. In the second scenario considered, the CA is not allowed to know the node's secret keys and N_i's private key and public key can be generated as follows. First, the node generates a random value v_i and submits $\mathbf{W}_i = v_i \times \mathbf{G}$ to the CA. Next, the CA generates a random h_i and calculates $h_i \times \mathbf{G}$. The CA then generates the pair of private and public keys by performing:

$$\begin{aligned} \mathbf{U}_i &= \mathbf{W}_i + h_i \times \mathbf{G} \\ p_i &= [H(ID_i, \mathbf{U}_i) \times h_i + d] \ mod\ ord\mathbf{G} \end{aligned} \quad (2)$$

and issues the values p_i and $\mathbf{U}_i$ to N_i. At this point, N_i generates his secret key $x_i = [p_i + H(ID_i, \mathbf{U}_i) \times v_i] \ mod\ ord\mathbf{G}$ and N_i can establish the validity of the values p_i and $\mathbf{U}_i$ issued to him by checking whether $p_i \times \mathbf{G} = H(ID_i, \mathbf{U}_i) \times (\mathbf{U}_i - \mathbf{W}_i) + \mathbf{R}$. Two important points should be noted here. First, in both cases $x_i \times \mathbf{G} = H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}$, and second since $x_i = [H(ID_i, \mathbf{U}_i) \times (h_i + v_i) + d] \ mod\ ord\mathbf{G}$, $x_i \times \mathbf{G} = H(ID_i, \mathbf{U}_i) \times U_i + \mathbf{R}$, which is identical to the case of the CA being allowed to know the node's secret keys.

C. Self-Certified DH Ephemeral Key-Generation

Two key generation procedures are commonly recognized: *fixed* and *ephemeral*. A fixed key-establishment procedure pertains to the case where two specific nodes use the same secret value (private key) whenever they wish to establish a joint key. In ephemeral key establishment, the two nodes generate a different key for each session, based on a random component invoked at each node. Ephemeral key-establishment is more secure and is generally preferred in many applications.

A self-certified DH ephemeral key-generation can be achieved by the following steps: (1) N_i and N_j generate a random 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 $\mathbf{EV}_j = pv_j \times \mathbf{G}$ (performed prior to establishing the communication session between the two nodes), (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 Nj generate the ephemeral session key,

$$\begin{aligned} K_{ij} \text{ (generated by } N_i) = pv_i \times [H(ID_j, \mathbf{U}_j)] \times \mathbf{U}_j + \mathbf{R}] \\ + (x_i + pv_i) \times \mathbf{EV}_j \end{aligned} \quad (3)$$

$$\begin{aligned} K_{ji} \text{ (generated by } N_j) = pv_j \times [H(ID_i, \mathbf{Ui})] \times \mathbf{U}_i + \mathbf{R}] \\ + (x_j + pv_j) \times \mathbf{EV}_i \end{aligned} \quad (4)$$

The two keys are expected to be identical, having the value $pv_i \times x_j \times \mathbf{G} + x_i \times pv_j \times \mathbf{G} + pv_i \times pv_j \times \mathbf{G}$. (i.e., N_i calculates: $pv_i \times [H(ID_j, \mathbf{U}_j)] \times \mathbf{U}_j + \mathbf{R}] + (x_i + pv_i) \times \mathbf{EV}_j = pv_i \times x_j \times \mathbf{G} + x_i \times pv_j \times \mathbf{G} + pv_i \times pv_j \times \mathbf{G}$). Similar logic is applied by the calculations performed at N_j. To complete the authentication cycle key confirmation needs to be preformed.

IV. DoS Mitigation and Ephemeral Key-Generation

The procedure for key generation described above does not include any mechanism for DoS mitigation. The DoS attack considered occurs when a malicious node repeatedly approaches legitimate nodes, requesting to establish a joint secret key. The energy consumed by the legitimate nodes, in the process of key generation, is substantial. Therefore, such an

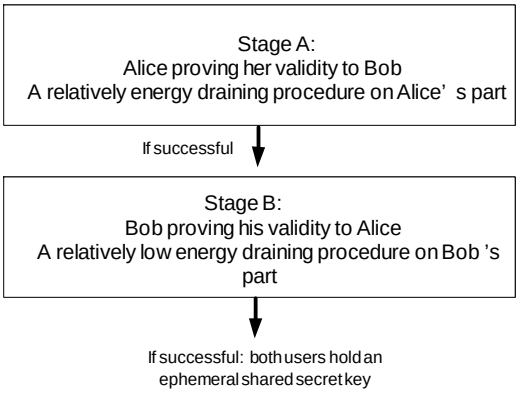

Fig. 1. The proposed procedure for Denial of Service (DoS) mitigation and ephemeral key-generation.

attack strategy can drain their energy. An efficient DoS mechanism should be able to mitigate such attacks. The proposed DoS mitigation approach comprises of two complementing parts. The first pertains to the instigator, Alice, who has to prove her validity to Bob, the party (node) approached. We assume that Alice is a node having limited resources similar to those of Bob. The second part, which takes into 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 identity of both Alice and Bob is 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 are the chances of a DoS attack diminished, but a different secret key will be generated. Figure 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 the previous section, 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.

A. The Instigator Node Proving Its Validity

The specific scenario described in this case pertains to a malicious node who 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 which 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 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 [30] with $e = 3$. We note that $e = 3$ is considered sufficiently secure [31]. (Higher levels of security are satisfied by $e = 2^{16}+1 = 65537$.) 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. 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. (For $e = 2^{16} + 1$ Bob has to calculate 17 modular multiplications.)

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 (who needs to calculate 2 modular multiplications, or 17 in the worse case), Alice has to perform a computationally heavy task as d_A typically consists of either 512 or 1024 bits. In the latter case she has to calculate 1536 modular multiplications, on the average, using the common square-and-multiply process. To that end, the approach proposed shifts the computational burden on the instigating node.

Step 4 - Bob compares the binary vector x he receives

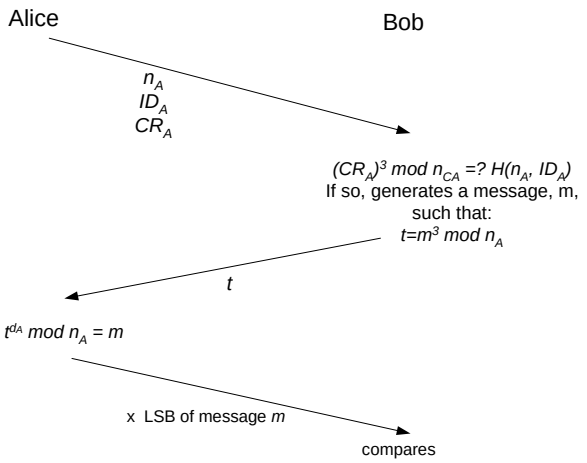

Fig. 2. DoS mitigation, based on Key Transport procedure.

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. It should be noted that this is achieved by performing 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. As described in [32] there is a need for only two key generations per node. 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 only twice for both key generations performed [32]. Finally, if Alice tries 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 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. 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?

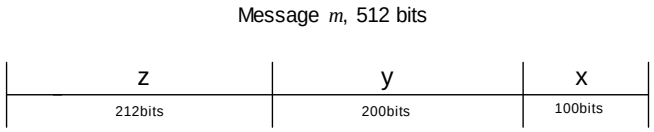

Fig. 3. Depicting a scenario where the original message is 512 bits.

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.

B. The Approached Node Proving It's Validity

If the first part of the procedure is successful, i.e., 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 key transport, (2) using the Elliptic Curve Digital Signature Algorithm (ECDSA) [33], and (3) using self-certified fixed key generation [32], [28], [34]. 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.

1) Key Transport: Bob can validate itself to Alice by using the RSA key transport method, similar to that described in section IV-A. 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 3). The subsequent 212 bits of message m will be labeled z. (The lengths of the components in the message can be negotiated between the two parties.)

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 un-

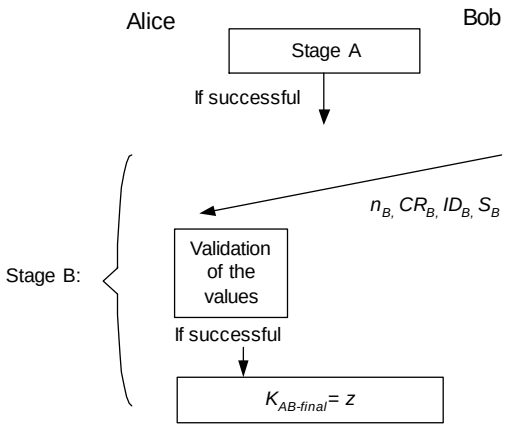

Fig. 4. Ephemeral key generation and denial of service mitigation using key transport.

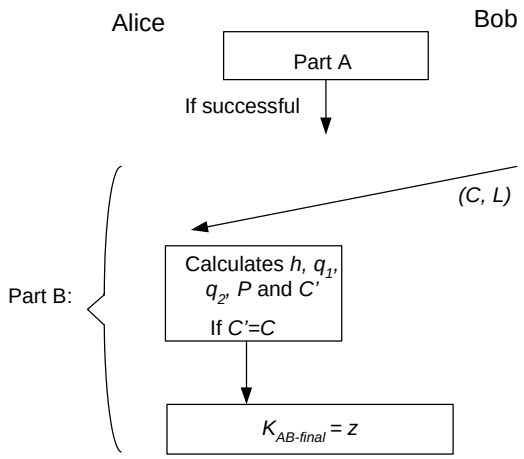

Fig. 5. Ephemeral key generation and denial of service mitigation using ECDSA.

deniably linked. Since $e = 3$, Alice computes only 2 modular multiplications. To check the validity of the certificate, Alice checks the following two equalities

$$(CR_B)^e \bmod n_{CA} \stackrel{?}{=} H(n_B, ID_B) \quad (5)$$

$$(S_B)^e \bmod n_B \stackrel{?}{=} y \quad (6)$$

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 MSB of message m. Figure 4 illustrates the complete process. To complete the authentication cycle key confirmation needs to be preformed.

2) Elliptic Curve Digital Signature Algorithm (ECDSA): Bob can also validate himself to Alice by using ECDSA 5. The latter is a method for digital signatures, based on ECC. The elliptic curve employed by the ECDSA can be the same one used in all procedures above. 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 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: $\mathbf{P} = q_1 \cdot \mathbf{G} + q_2 \cdot \mathbf{V}$, where n_B is Bob's public key, and calculates $C\prime$, the scalar representation of point P. The algorithm concludes when Alice validates that $C = C\prime$. If the latter holds, Bob is validated.

Step 3 - The ephemeral key resulting will be denoted by $K_{AB-final} = z$, corresponding to the MSB of message m. Figure 5 illustrates the complete process. To complete the authentication cycle key confirmation needs to be preformed.

3) Self-Certified DH Fixed Key-Generation: One of the methods in which Bob can prove his validity to Alice is by using a self certified method similar to the ephemeral one described in section III-C. The ephemeral method can certainly be used, but when the primary focus is to minimize energy drainage, a self certified fixed key generation is advisable since it consists of less computations. We now go back to the notations used in section III where self certified ephemeral key generations were described.

A self-certified DH fixed key-generation is achieved by the following two steps [28] : (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,

$$\begin{aligned} K_{ij} \text{ (generated by } N_i) &= x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] \\ K_{ji} \text{ (generated by } N_j) &= x_j \times [H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}]. \end{aligned} \quad (7)$$

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

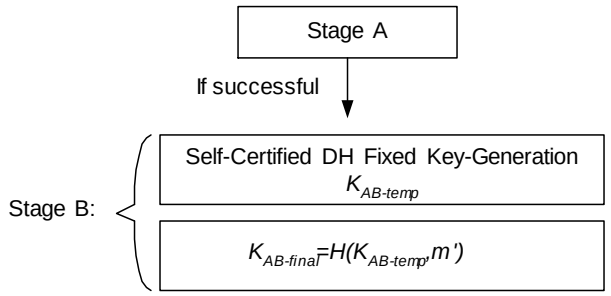

Fig. 6. Ephemeral key generation and denial of service mitigation using a self-certified DH fixed key-generation.

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 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 ([28]), 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

$$K_{AB-final} = H(K_{AB-temp}, m\prime), \tag{8}$$

where H is a hash function and $m\prime$ is the random message m, excluding the x least significant bits (see figure 6).

V. Implementation Results

As discussed in section III-C, a self certified ephemeral key generation can be generated without DoS mitigation. We have also shown in section IV-B three different methodologies for generating an ephemeral key for which the DoS mitigation was an embedded component. This section presents implementation results pertaining to all three methods, providing a comparison in terms of timing and energy resources.

The methodology described in stage A and all of the three methodologies discussed in stage B were implemented on the Intel Mote 2 [35] 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. The self-certified algorithms (both fixed and ephemeral) were implemented using functions taken from the TinyECC package [23]. 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 ephemeral key generation, excluding DoS mitigation, takes 102 msec to complete, and consumes 46 mJ at each node. Since all computations are symmetric, the entire process takes 204 msec to complete and requires 92 mJ.

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, results that substantiated 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 in section IV-B.2, Bob preforms one point by scalar multiplication while Alice performs two. When using the self certified fixed key method, each of the nodes perform one point by scalar multiplication. Each multiplication takes 50 msec and 23.16 mJ. Please see tables I and II 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 as 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 method in both stages are almost symmetric, when it comes to the computational load that Bob and Alice have.

	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	100	50	46.32	23.16	150	69.48
Fixed Key	50	50	23.16	23.16	100	46.32

TABLE I

TIME (MSEC) AND ENRGY (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 consumption	Both stages	Both stages
Key Transport	463.06	213.01
ECDSA	381.02	175.75
Fixed Key	331.02	152.6

TABLE II

TOTAL TIME (MSEC) AND ENRGY (MJ) CONSUMED WHILE PERFORMING BOTH STAGES A AND B FOR 1024-BIT RSA AND 160-BIT ECC ON THE INTEL MOTE 2 PATFORM FOR 312 MHZ CORE CLOCK

At first glance, it may appear that this symmetry attribute can increase the lifetime of the network. However, due to the specific key generation process proposed in [32], it is clear that by the ad-hoc nature of WSN clusters, all nodes have an equal likelihood of being instigators. Hence this symmetry attribute is of no particular significance. When comparing ECDSA to fixed key generation, we first come to the conclusion that the fixed key method is more efficient since it implies a 15% energy gain and reduced time.

Generating a self-certified ephemeral key consumes an estimated 200 msec and 92 mJ. It can be observed from table II that embedding DoS mitigation, while generating the ephemeral key, increases the time and energy in the fixed-key case by 65%. This is the cost associated by incorporating DoS mitigation in the ephemeral key generation process.

VI. CONCLUSIONS

This paper introduced a public key cryptographic method for preventing DoS attacks that target the draining of battery energy in WSN ephemeral key establishment. By exploiting the asymmetry in RSA signature generation, a robust approach to minimizing energy usage at the node being attacked has been proposed. Combining the DoS mitigation with self-certified ECC-based key generation yielded a highly resource-efficient security framework. Moreover, the concept developed can be applied to a wide range of additional security services that are currently not offered in WSN environments. Implementation results using the Intel Mote 2 platform indicate that the methodology is practical and efficient.

VII. ACKNOWLEDGEMENTS

This research is sponsored in part by NSF grant ECS-0449309 (CAREER). The authors would like to thank Itamar Elhanany at the University of Tennessee and Peng Ning at North Carolina State University for the insightful discussions pertaining to this paper.

REFERENCES

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

[2] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, pp. 53–57, June 2004.

[3] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "Tinypk: Securing sensor networks with public key technology," in *Proc. of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, pp. 59–64, 2004.

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

[5] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient routing protocols for wireless microsensor networks," in *Proc. 33rd Hawaii Intt'l Conf. on System Sciences (HICSSt'00)*, pp. 3005–3014, January 2000.

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

[7] 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, pp. 1172–1183, August 2003.

[8] 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, (Newport Beach, CA), pp. 54–61, July 2004.

[9] W. Du, J. Deng, Y. S. Han, S. Chen, and P. K. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge," in *Proc. of IEEE INFOCOM 2004*, (Hong Kong, China), 2004.

[10] W. Zhang and G. Cao, "Group rekeying for filtering false data in sensor networks: A predistribution and local collaboration-based approach," in *Proceedings of the 2005 IEEE INFOCOM*, (Miami, FL, USA), 2005.

[11] 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), pp. 42–51, October 2003.

[12] A. Chan, "Probabilistic distributed key pre-distribution for mobile and ad hoc networks," in *Proceedings of the 2004 IEEE International Conference on Communications*, pp. 3743–3747, June 20-24 2004.

[13] M. Ramkumar and N. Memon, "An efficient key predistribution scheme for ad hoc networks security,"

[14] 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), pp. 41–47, November 2002.

[15] 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*, Oct 2004.

[16] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Proceedings of the third IEEE International Conference on Pervasive Computing and Communication (PerCom 2005)*, pp. 324–328, 2005.

[17] B. Arazi, I. Elhanany, O. Arazi, and H. Qi, "Revisiting public key cryptography for wireless sensor networks," *IEEE Computer Magazine*, pp. 85–87, November 2005.

[18] E. O. Blaβ and M. Zitterbart, "Towards acceptable public-key encryption in sensor networks," in *The 2nd International Workshop on Ubiquitous Computing*, ACM SIGMIS, May 2005.

[19] Q. Huang, J. Cukier, H. Kobayashi, B. Liu, and J. Zhang, "Fast authenticated key establishment protocols for self-organizing sensor networks," in *International Conference on Wireless Sensor Networks and Applications*, 2003.

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

[21] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*. Boston, MA: Kluwer Academic Publishers, 1993.

[22] "Mica2 datasheet." Crossbow Technology, Inc. Available at http://www.xbow.com.
[23] P. Ning and A. Liu, "TinyeCC: Elliptic curve cryptography for sensor networks," tech. rep., 2005. http://discovery.csc.ncsu.edu/software/TinyECC.
[24] M. Girault, "Self-certified public keys," in *Advances in Cryptology–EUROCRYPT'91*, pp. 491–497, March 1991. LNCS - Springer-Verlag.
[25] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology (CRYPTO '86)*, vol. 263, pp. 186–196, March 1987.
[26] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient protocol for authenticated key agreement," Tech. Rep. CORR 98-05, 1998.
[27] "Fact sheet nsa suite b cryptography." http://www.nsa.gov, 2005.
[28] 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.
[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, pp. 816–824, Novemeber 2002.
[30] 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.
[31] J. Jonsson and B. Kaliski, "Public-key cryptography standards (pKCS) #1: RSA cryptography specifications version 2.1," 2003.
[32] O.Arazi and H. Qi, "Load-balanced key establishment methodologies in wireless sensor networks," *International Journal of Sensor Networks, IJSN*, vol. 1, April 2006.
[33] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Designs, Codes and Cryptography*, vol. 19, pp. 173–193, 2000.
[34] B. Arazi, "Certification of dl/ec keys," in *Proc. of the IEEE P1363 Study Group for Future Public-Key Cryptography Standards*, May 1999.
[35] R. Adler, M. Flanigan, J. Huang, R. Kling, N. Kushalnagarand, L. Nachman, C. Y. Wan, and M. Yarvis, "Intel mote 2: an advanced platform for demanding sensor network applications," in *SenSys ?05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 292–298, 2005.