Toward Mitigating Denial of Service Attacks in Power-Constrained Sensor Networks

O. Arazi[1,2], H Qi[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. 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 present an 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.**

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]. This is particularly true in health and military applications, where critical information is frequently exchanged among sensor nodes through insecure wireless media. 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. A fundamental requisite for security, other than providing data confidentiality and authentication, is Denial of Service (DoS) prevention. The computational effort involved in performing 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. To address this issue, we present a public key cryptographic approach for mitigating DoS attacks in sensor networks.

II. 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 [3]. 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.

III. DoS Mitigation and Key-Generation

The ECC-based procedure for key generation, which included certification, as described by the authors in [4] 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 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. We shall refer to the following notations in the context of the proposed DoS mitigation scheme: n_i is user i's public key, d_i his private key, CR_i his (CA issued) certificate, and ID_i his public key identification. 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 [5] using RSA [6] with $e = 3$. We note that $e = 3$ is considered sufficiently secure. (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 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 [4] 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 [4]. 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.

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? 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), and (3) using self-certified fixed key generation [4], [7]. 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 III-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. 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 undeniably linked. Since $e = 3$, Alice computes only 2 modular multiplications. To check the validity of the certificate, Alice checks the following two equalities

$$(CR_B)^e \bmod n_{CA} \stackrel{?}{=} H(n_B, ID_B) \tag{3}$$

$$(S_B)^e \bmod n_B \stackrel{?}{=} y \tag{4}$$

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 MSB of message m. 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. 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. 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. 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 II where self certified ephemeral key generations were described.

A self-certified DH fixed key-generation 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,

$$\begin{aligned} K_{ij} \text{ (generated by } N_i) &= x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] \\ K_{ji} \text{ (generated by } N_j) &= x_j \times [H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}]. \end{aligned} \tag{5}$$

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 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, 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

$$K_{AB-final} = H(K_{AB-temp}, m\prime), \tag{6}$$

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

IV. 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.

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] 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.

[4] O.Arazi and H. Qi, "Load-balanced key establishment methodologies in wireless sensor networks," *International Journal of Sensor Networks, IJSN*, vol. 1, April 2006.

[5] 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.

[6] 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.

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