Self-Certified Public Key Generation on the Intel Mote 2 Sensor Network Platform

O. Arazi, I. Elhanany[1], D. Rose, H. Qi
Electrical & Computer Engr. Dept.
University of Tennessee
e-mail: {oarazi,itamar,derek,hqi}@utk.edu
[1]corresponding author

B. Arazi
Computer Science & Computer Engr. Dept.
University of Louisville
e-mail: ben.arazi@louisville.edu

***Abstract*— It is widely acknowledged that security will play a key role in the successful design and deployment of wireless sensor networks (WSN). A prerequisite for achieving security is the ability to dynamically establish a secret key joint to two nodes. Elliptic Curve Cryptography (ECC) has emerged as a suitable public key cryptographic foundation for WSN. This poster will present an efficient ECC-based method for self-certified key generation in resource-constrained sensor nodes. In particular, we provide implementation results on the Intel Mote 2 sensor network platform which demonstrate that such key generation can be established in the order of 60 msec while consuming less than 30 mJ.**

I. Introduction

Recent advancements in the design and fabrication of low-power VLSI circuitry, as well as wireless communications, have broadened the applications prospect for wireless sensor networks (WSN). The latter promise to revolutionize our ability to sense and control diverse physical environments using large numbers of small, inexpensive devices that integrate sensing, computation and communication. Many WSN applications, spanning military and civilian, assume that sensor nodes are deployed in hostile environments where they are prone to a wide variety of malicious attacks. As a result, security becomes a key concern [1].

Although a variety of key-generation methods have been proposed for WSN, they cannot be directly transplanted in sensor network environments. A simple solution for key establishment is a single network-wide shared key. Unfortunately, a single node in the network being captured would easily reveal the network secret key. Therefore, a current mainstream effort consists of random key pre-distribution, 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 [2][3]. These schemes offer partial solution with respect to scalability, cryptographic robustness and the ability to append and revoke security attributes. The necessity for public key cryptographic key generation in WSN is widely acknowledged. Public key cryptography offers a higher degree of scalability and decentralized management, both of which are strongly coherent with the ad-hoc nature of WSN. Elliptic Curve Cryptography (ECC) has emerged as a suitable public key cryptographic foundation for sensor networks, providing high security for relatively small key sizes. Recent results [4] indicate that the execution of ECC operations in sensor nodes is feasible, with predictable improved performance.

This poster will describe a scalable resource-efficient key generation algorithm and protocol, specifically optimized for WSN. In particular, we will present implementation results, including timing and energy consumption, on the Intel Mote 2 platform, which strongly suggest that public key cryptography is a pragmatic infrastructure for next-generation WSN security solutions.

II. Cryptographic Model and Goals

This poster addresses the need for resource-efficient *fixed* as well as *ephemeral* public key-generations. The former relates to the case where two specific nodes generate the same secret value whenever they wish to establish a joint key. In ephemeral key-generation, the two nodes generate a different key for each session established, based on a random component introduced by each node. Ephemeral key-generation is more secure and is generally preferred when time and resources permit. The core of ECC-based key generation comprises of point by scalar multiplication operations. These consume the majority of the energy involved in the key establishment process. As an example, transmitting 1 kbit of data consumes two orders of magnitude less energy than performing a single point by scalar multiplication operations.

In a public cryptographic session, a need emerges to authenticate the public values submitted by the participants. Customarily, this is facilitated by the use of a certificate, issued by a Certifying Authority (CA), attesting to the connection between a user's public key and his ID. Verifying the authenticity of certified values requires a reference to the public key of the CA. An authentication procedure which is based on certification therefore needs the following values as input: the user's public key, his ID, the certificate and the CA's public key.

In self-certified public key cryptographic applications [5][6], a user submits its ID along with its public key, but does not submit an explicit certificate, thereby reducing communication and management overheads, which is a vital consideration in WSN. Verifying the validity of a user's public key, that is, verifying that the public key is associated with the user's ID, is achieved in an implied manner that still needs an explicit reference to the CA's public key. In identity-based systems, the user's public key is its actual ID, which removes 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.

III. Efficient Self-Certified Diffie Hellman (DH) Key Generation

A. Notation and Formulation

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 DH key generation, we must first define some notation and terminology. 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 poster: G – a generating group-point, used by all relevant nodes; $ordG$ – the order of G. (exponents are calculated $modulo\ ordG$); d – the CA's private key; $\mathbf{R}$ – the CA's public key (where $\mathbf{R} = d \times G$); 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 in this section 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 G, it 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 $h_i \times G$**.** Next, the CA then generates node i's public and private keys by performing:

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

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 $x_i \times G = H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}$. 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 $\mathbf{W}_i = v_i \times G$ to the CA. Next, the CA generates a random h_i and calculates $h_i \times G$. The CA then generates the pair of private and public keys by performing:

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

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 $p_i \times G = H(ID_i, \mathbf{U}_i) \times (\mathbf{U}_i - \mathbf{W}_i) + \mathbf{R}$. Two important points should be noted here. First, in both cases $x_i \times G = H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}$, and second since $x_i = [H(ID_i, \mathbf{U}_i) \times (h_i + v_i) + d]\ mod\ ordG$, $x_i \times G = H(ID_i, \mathbf{U}_i) \times U_i + \mathbf{R}$, which is identical to the case of the CA being allowed to know the node's secret keys.

C. Fixed Key-Generation

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,

$$\begin{aligned} &K_{ij} \text{ (generated by } N_i) = x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] \\ &K_{ji} \text{ (generated by } N_j) = x_j \times [H(ID_i, \mathbf{U}_i) \times \mathbf{U}_i + \mathbf{R}]. \end{aligned} \tag{3}$$

The two keys are expected to be identical, having the value $x_i \times x_j \times \mathbf{G}$. (i.e. N_i calculates: $x_i \times [H(ID_j, \mathbf{U}_j) \times \mathbf{U}_j + \mathbf{R}] = x_i \times [H(ID_j, \mathbf{U}_j) \times h_i \times G + d \times G] = x_i \times [H(ID_j, \mathbf{U}_j) \times h_i + d\ ] \times G = x_i \times x_j \times \mathbf{G}$. 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.

D. Ephemeral Key-Generation

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 $\mathbf{EV}_i = pv_i \times G$, N_j calculates the ephemeral value $\mathbf{EV}_j = pv_j \times G$ (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,

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

$$K_{ji} \text{ (generated by } N_j) = pv_j \times [H(ID_i, \mathbf{Ui})] \times \mathbf{U}_i + \mathbf{R}] + (x_j + pv_j) \times \mathbf{EV}_i \quad (5)$$

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

IV. Implementation Results

The methodologies developed were implemented on the Intel Mote 2 [7] 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 [8]. The latter targeted the MICAz 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 code was written in NesC running on the TinyOS operating system. Nodes exchange messages using a 2.4 GHz embedded low-power radio transceiver.

Figure 1 depicts the results obtained for 160-bit ECC-based fixed and ephemeral key generations, as a function of the CPU clock frequency used. It should be noted that 160-bit keys in ECC are equivalent, from a cryptocomplexity perspective, to 1024-bit keys in RSA. 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 is demonstrated for operating at higher frequencies. At a frequency of 312 MHz, for example, fixed key generation is achieved in less than 50 msec, consuming 23 mJ. This is compared to approximately 7 seconds and 300 mJ, when executing similar ECC operations running on the 7.37 MHz/8-bit MICAz mote (as reported in [8]). As depicted in figure 1, the methodology proposed is highly practical, paving the way for broader development of resource-efficient security mechanisms for wireless sensor networks.

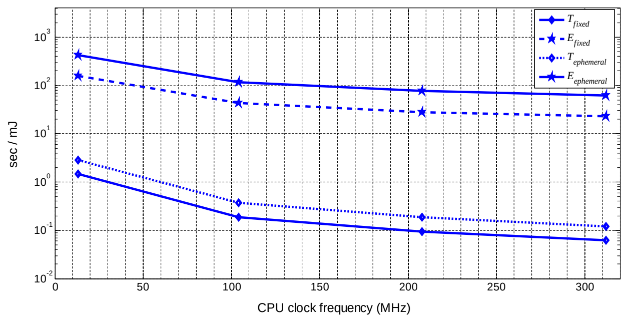

Fig. 1. Time and energy measurements for ECC-based 160-bit key generation on the Intel Mote 2 platform.

V. Summary and Future Directions

This work presented an efficient technique for ECC-based public key generation in wireless sensor networks. Novel algebraic derivations were described, addressing the need for both fixed and ephemeral key generations. Efficient implementations on the Intel Mote 2 sensor network platform suggest that the time is ripe for public key cryptographic security be introduced in WSN applications.

Future work will focus on extending the approach to address group key generations, such as those required by ad-hoc clusters. Moreover, the issue of denial of service (DoS) will be a primary area of future research. In the context of public key cryptography, DoS attacks in WSN typically emerge when malicious nodes attempt to drain the energy source of legal nodes by continuously requesting the latter to generate session keys.

References

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

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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 '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 292–298, 2005.

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