An Efficient Scheme for Authenticating Public Keys in Sensor Networks

Wenliang Du* and Ronghua Wang
Department of Electrical Engineering
and Computer Science
Syracuse University
{wedu, rwang01}@ecs.syr.edu

Peng Ning†
Department of Computer Science
North Carolina State University
pning@ncsu.edu

ABSTRACT

With the advance of technology, Public Key Cryptography (PKC) will sooner or later be widely used in wireless sensor networks. Recently, it has been shown that the performance of some public-key algorithms, such as Elliptic Curve Cryptography (ECC), is already close to being practical on sensor nodes. However, the energy consumption of PKC is still expensive, especially compared to symmetric-key algorithms. To maximize the lifetime of batteries, we should minimize the use of PKC whenever possible in sensor networks.

This paper investigates how to replace one of the important PKC operations–the public key authentication–with symmetric key operations that are much more efficient. Public key authentication is to verify the authenticity of another party's public key to make sure that the public key is really owned by the person it is claimed to belong to. In PKC, this operation involves an expensive signature verification on a certificate. We propose an efficient alternative that uses one-way hash function only. Our scheme uses all sensor's public keys to construct a forest of Merkle trees of different heights. By optimally selecting the height of each tree, we can minimize the computation and communication costs. The performance of our scheme is evaluated in the paper.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols, Wireless Communications

General Terms

Algorithms, Design, Security, Performance

*Du's work was supported by Grants ISS-0219560 and CNS-0430252 from the United States National Science Foundation.
†Ning's work was supported by Grants CNS-0430223 and CAREER-0447761 from the United States National Science Foundation.

MobiHoc'05, May 25–27, 2005, Urbana-Champaign, Illinois, USA.

Keywords

Wireless sensor networks, public key, merkle tree, deployment knowledge

1. INTRODUCTION

Sensor networks are being deployed for a wide variety of applications, including military sensing and tracking, environment monitoring, patient monitoring and tracking, smart environments, etc. When sensor networks are deployed in a hostile environment, security becomes extremely important as these networks are prone to different types of malicious attacks. For example, an adversary can easily listen to the traffic, impersonate one of the network nodes, or intentionally provide misleading information to other nodes. To provide security, communication should be encrypted and authenticated. A challenge is how to bootstrap secure communications between sensor nodes, i.e., how to set up secret keys between communicating nodes. This problem is known as the *key agreement* problem.

Symmetric key techniques are attractive for this task due to their energy efficiency. A number of techniques have been proposed recently [4, 6, 9, 12, 19, 27]; their basic idea is that the secret keys are pre-distributed among sensors before their deployment. Their goal is to use the least amount of memory to achieve the highest level of connectivity (i.e., high percentage of the neighboring sensors should be able to establish secure communications between them) and the highest level of resilience (i.e., the capture of some sensors by an adversary should not jeopardize the security of other sensors). However, due to the limitation on memory, these techniques are not able to achieve both a perfect connectivity and a perfect resilience for large-scale sensor networks.

The use of Public-Key Cryptography (PKC) would eliminate the above problem. Because of its asymmetry property, sensors do not need to carry the pre-distributed keys. Any two sensors can establish a secure channel between themselves, and the capture of some sensors will not affect the security of others. PKC is widely used in the Internet for bootstrap secure communication. For example, SSL (Secure Socket Layer) and IPsec standards both use PKC for their key agreement protocols.

The most common criticism on using PKC in sensor networks is its computational complexity and communication overhead. Recently, a number of studies have been conducted to find out how practical it is to use PKC for sensor networks [14, 15, 21]. Their results show that PKC is indeed feasible to be used in sensor networks. For example, Gura et al. [15]. show that Elliptic Curve Cryptography (ECC) signature verification takes $1.62s$ with 160-bit keys on ATmega128 8MHz processor, a processor used for Crossbow motes platform [7]. These results indicate that, with the ad-

vance of fast growing technology, it will not be long before public key cryptography becomes widely accepted for wireless sensor networks.

Even with PKC getting faster and faster, performance difference between PKC and symmetric key cryptography is not going to change much unless some breakthrough in PKC occurs. Compared to the symmetric key cryptography, the cost of PKC is still much more expensive. For instance, a 64-bit RC5 encryption on ATmega128 8MHz takes 5.6 milliseconds, and a 160-bit SHA1 hash function evaluation takes only 7.2 milliseconds [13]. These are more than 200 times faster than PKC algorithms, and the gap is unlikely to disappear. Furthermore, public key cryptography is not only expensive in computation, but also in communication. For instance, to send a public key from one node to another using RSA [25], at least 1024 bits needs to be sent if the private key is 1024 bits long. Therefore, even after PKC is implemented in sensor nodes, we still need to treat PKC as expensive operations, and we need to use it more selectively and efficiently in order to maximize the lifetime of sensor networks.

A number of studies have been focusing on improving the efficiency of PKC. They primarily focus on optimizing the implementation of PKC algorithms for the processors of sensor nodes [14, 15, 21]. The optimization is achieved by exploring the hardware properties and architecture of sensor processors. In this paper, we take another approach toward optimization: *we explore network properties of sensor networks to reduce the amount of expensive public-key operations in PKC-based protocols*.

The objectives of the paper. This paper focuses on the optimization of an essential operation in PKC: the public key authentication. Before a node A uses the public key from another node B, A must verify that the public key is actually B's[1], i.e., A must authenticate B's public key; otherwise, the man-in-the-middle attack is possible. In general networks, public key authentication is achieved by certificates. Namely, A verifies a certificate of B's public key, and the certificate is signed by a certificate authority (CA), whom A and B both trust. The certificate approach can also be implemented in sensor networks: let all the sensors carry the CA's public key, so they can use this key to verify signatures on certificates.

Since authenticating public key is essential for PKC, and it is likely to be used for many times, it is important to optimize this operation for the power-constrained sensor networks. We focus on this public key authentication optimization problem in this paper. Our goal is to find a scheme to verify the authenticity of public keys using algorithms that consume much less energy than PKC.

Overview of our approach. We propose to use one-way hash function to conduct public key authentication in sensor networks. An important difference between sensor networks and general networks is that sensors usually belong to one administrative entity before their deployment. Therefore, they can exchange the one-way hash values of their public keys securely prior to the deployment. The Merkle tree technique can be used such that each sensor only needs to save one hash value while being able to authenticate all the public keys. However, since the communication overhead required by the authentication operation is proportional to the height of the Merkle tree, the overhead can be quite high for large-size sensor networks.

To reduce the communication overhead, we trim down the single Merkle tree to a number of shorter trees, which do not need to have the same height. To minimize the communication overhead, we formulate the following problem: *Given that a sensor can store up to m hash values, how to trim a Merkle tree, such that the overall communication overhead is minimized?*

We developed a trimming scheme based on sensor deployment knowledge. Our basic idea is the following: if B is more likely to be A's neighbor, B should be in a shorter tree of A, so the communication overhead to authenticate B's public key is lower; if B is less likely to be A's neighbor, B can be put in a taller tree of A, because the authentication is less likely to occur. We have evaluated the performance of our schemes. The results show that our scheme can save up to 86% of the energy for the public key authentication operation.

Organization. The rest of the paper is organized as follows. Section 1.1 discuss related work. Section 2 describes a basic solution to the public key authentication problem. Section 3 presents an improved scheme, which uses deployment knowledge to reduce the communication and computation overhead. Section 4 presents our evaluation results. Finally we draw the conclusion and lay out our future work in Section 5.

1.1 Related Work

There are extensive studies on using symmetric-key cryptography to achieve various aspects of security in sensor networks. Perrig et al. developed a security architecture for sensor networks, which includes SNEP(Security Network Encryption Protocol), a security primitive building block,and μTESLA [23]. Liu and Ning proposed a multi-level key chain method for the initial commitment distribution in μTESLA [18]. Karlof, Sastry and Wagner developed TinySec, the first fully implemented link layer security architecture for sensor networks [16].

To achieve efficient key management, several symmetric key-based techniques were proposed in the past [4–6, 9, 12, 19, 27]; Carman, Kruus, and Matt studied the performance of a number of key management approaches in sensor network on different hardware platforms [5]. Eschenauer and Gligor [12] proposed a probabilistic key pre-distribution technique to bootstrap the initial trust between sensor nodes. Chan et al. further extended this idea and proposed the q-composite key pre-distribution [6]. This approach allows two sensor nodes to set up a pairwise key only when they share at least q common keys. Chan et al. also developed a random pairwise keys scheme to defeat node capture attacks. Du et al. developed a pairwise key management scheme [9], which was also independently discovered by Liu and Ning [19]. Zhu et al. proposed a protocol suite named LEAP to help establish individual keys between sensors and a base station, pairwise keys between sensors, cluster keys within a local area, and a group key shared by all nodes [27].

To fully take advantage of the information available to the sensor networks, schemes using information from the environment were proposed. Deployment knowledge from the environment is one frequently used. For example, Du et al. proposed a key management scheme using deployment knowledge [10]. Liu et al. proposed location-based pairwise key establishment for relatively static sensor networks [20]. Huang et al. further proposed a grid-group scheme which uses known deployment information in [8]. With the apriori knowledge obtained before distributing the sensor nodes, the memory usage of the sensor nodes is greatly improved while the connectivity of the sensor network is maintained.

In addition to the studies on symmetric key cryptography, recently, there are a number of studies investigating the implementation of PKC in sensor networks [14, 15, 21, 26]. These studies have been focusing on measuring the performance of PKC algorithms on power-constrained processor and developing optimized implementation of PKC algorithms for sensor networks.

[1] PKC based on identity-based encryption [3] is an exception.

2. THE BASIC SCHEME

Essentially, the goal of the public key authentication is to make sure that the binding between an identify and a public key is authentic. The certificate approach is designed for users who do not have a pre-established trust relationship to be able to authenticate each other's public key. They achieve this using a third party, the certificate authority (CA), with whom they both have a trust relationship. However, if two users already have a trust relationship, it is not necessary to use certificates. For example, if two users have met before, they could exchange their public keys while they were physically together. Man-in-the-middle attacks will not be possible in this scenario.

In sensor networks, nodes indeed have "met" before their deployment because all these nodes usually belong to the same administrative entity. This is a major difference between sensor network environments and Internet environments. In many sensor-network applications, sensors do "know" and "trust" each other before the deployment. In other words, before their deployment, sensors are in a benign environment where they can exchange information in plaintext and thus establish trust relationships among themselves. With this benign pre-deployment phase that is unique to sensor networks, public key authentication after deployment can be achieved in a much more power-efficient way.

2.1 A Naive Scheme

A naive solution to the public key authentication problem without using certificates is to let each node carry all the other nodes' public keys. However, since the size of public keys can be large, sensor might not have enough memory to save all the public keys. We can improve the memory-usage situation by letting each node carry a one-way hash value of those public keys. When two nodes exchange their public keys, they just need to compute the one-way hash value of the received public keys and check whether the results match the values stored in their memory. Therefore, public key authentication is reduced to a one-way hash function evaluation, which consumes two to three order of magnitude less energy than a public-key operation.

The above naive solution still has the memory-usage problem. For a sensor network of size N, each sensor needs to devote $(N-1)L$ memory to those hashes, where L is the length of each hash value (e.g. $L = 160$ bits for SHA1 [11]). For example, when $N = 10000$, 195K bytes of memory are needed for SHA1. Sensors usually do not have that much of data memory. Merkle trees [22] can be used to solve the memory-usage problem.

2.2 A Memory-Efficient Scheme

The Merkle tree (also called hash tree) is a complete binary tree equipped with a function $hash$ and an assignment Φ, which maps a set of nodes to a set of fixed-size strings. In a Merkle tree, the leaves of the tree contain the data, and the Φ value of an internal tree node is the hash value of the concatenation of the Φ values of its two children.

Building Merkle Tree. To build a Merkle tree for our problem, we construct N leaves $L_1, \ldots, L_N$, with each leaf corresponding to a sensor node. Each leaf contains the bindings between the identity of its corresponding node and the public key of the node. We then build a complete binary tree with these leaves. The Φ value of each node is defined as the following (we use V to denote an internal tree node, and V_{left} and V_{right} to denote V's two children; we use id_i to represent node i's identity, and we use pk_i to represent node i's public key):

$$\begin{aligned} \Phi(L_i) &= hash(id_i, pk_i), \text{ for } i = 1, \ldots, N \\ \Phi(V) &= hash(\Phi(V_{left})||\Phi(V_{right})), \end{aligned}$$

where "||" represents the concatenation of two strings, and the function $hash$ is a one-way hash function such as MD5 or SHA1. Figure 1 depicts an example of the Merkle tree built for our purpose. Each sensor only needs to store $\Phi(R)$, where R is the root of the Merkle tree. The memory usage is the length of one hash value.

Authenticating Public Keys. Let pk be Alice's public key, and L be Alice's corresponding leaf node in the tree. Let λ denote the path from L to the root (not including the root), and let H represent the length of the path. For each tree node $v \in \lambda$, Alice sends $\Phi(v$'s sibling) to Bob, along with the public key pk. We use $\lambda_1, \ldots, \lambda_H$ to represent these Φ values, and we call these Φ values the proofs.

To verify the authenticity of Alice's public key pk (assume Alice's identity is id), Bob computes $hash(id, pk)$; he then uses the results and $\lambda_1, \ldots, \lambda_H$ to reconstruct the root of the Merkle tree R' with $\Phi(R')$. Bob will trust that the binding between id and pk is authentic only if $\Phi(R') = \Phi(R)$. An example is depicted in Figure 1. The solid dot circle in the figure represent the proofs for Alice. Because of properties of one-way hash functions, it is computationally infeasible for adversaries to forge proofs for a binding that is not one of the leaves of the Merkle tree [22].

Communication Costs. Although the Merkle tree can reduce the memory usage to just one hash value, this does not come for free: in the naive solution, Alice does not need to send extra information for the public key authentication (she has to send the public key anyway); however, in the improved solution, Alice needs to send all those proofs, which consists of H hash values, where H is the height of the tree. Because the Merkle tree is a complete binary tree with N leaves, its height is $\log N$ (the base of the logarithm is assumed to be 2 throughout this paper). Therefore, the communication costs is $L \cdot \log N$, with L being the length of a hash value.

Optimization can be made. Since each sensor stores up to $\log n$ hash values in its memory, two sensors might have stored some common hash values. A sensor does not need to send to another sensor the hash values that they share; communication costs can be reduced. However, the following theorem indicates that the saving is not substantial:

THEOREM 2.1. *The expected number of common hash values between two arbitrary sensor nodes is approximately* 1.

PROOF. We call a subtree of the Merkle tree a level-k subtree if the depth of the root of this subtree in the Merkle tree is k. For example, the direct subtree of the root is the level-1 subtree. For the two sensors to share only one hash value (excluding the root of the Merkle tree), they must be in the same level-1 subtree, but not in the same level-2 subtree; the probability of that is $\frac{1}{2} - \frac{1}{4} = \frac{1}{4}$. Similarly, for the two sensors to share exactly k hash values, they must be in the same level-k subtree, but not in the same level-$(k+1)$ subtree; the probability of that is $\frac{1}{2^k} - \frac{1}{2^{k+1}} = \frac{1}{2^{k+1}}$. Therefore, the expected amount of savings is the following

$$\sum_{i=2}^{\log N} \frac{i-1}{2^i} = 1 - \frac{1}{N} \approx 1.$$

□

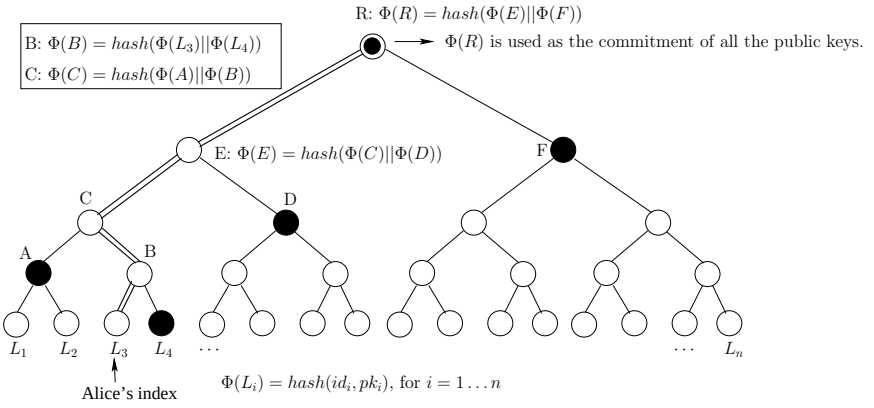


Figure 1: Using Merkle tree To Authenticate Public Keys.

Because of the $L \cdot \log n$ communication overhead, compared to the original certificate approach, the energy saving on the computation might not be able to make up for the extra energy consumption introduced by the communication overhead. Further improvement on the communication overhead must be made, and we will discuss this issue in the next section.

2.3 Improving Communication Overhead

The naive scheme and the above memory-efficient scheme are two extreme schemes: the naive scheme has the best performance on communication but the worst performance on memory usage, while the memory-efficient scheme is the opposite: it has the best memory usage (only needs to store one hash value), but a much worse communication overhead. A compromise can be made between these two extreme schemes. For example, we can increase the amount of memory usage to reduce the communication overhead.

A straightforward compromise is to trim down the Merkle tree constructed in the memory-efficient scheme, and thus turn the single Merkle tree into a number of sub-trees; these sub-trees are still Merkle trees. We call the initial Merkle tree the *original* Merkle tree; we call the sub-trees the Merkle *sub-trees*, and we call the set of these Merkle sub-trees a *Merkle forest*.

Each sensor now carries the roots of all the trees in the Merkle forest. To authenticate a public key, one just uses the public key's corresponding Merkle tree. Because the height of each tree is now smaller than the original Merkle tree, the communication overhead is reduced.

For example, if we remove only the root of the Merkle tree in the basic scheme, we get two smaller Merkle trees; the communication costs is reduced by one hash value. In general, if we remove k levels of the original Merkle tree, the communication costs is reduced by k hash values. However, this gain does not come for free; on one hand, the communication is reduced, but on the other hand, the memory usage increases to 2^k hash values. Since sensors are usually memory constrained, the value of k is quite limited. Given that only m hash values can be stored in the memory, the average communication overhead is about $(\log \frac{N}{m})$.

As we can see, the above straightforward trimming method is not optimal, and we should find some optimized way to trim down the Merkle tree. In the next section, we will describe a further improvement on the basic scheme to minimize the overall communication overhead.

3. AN IMPROVED SCHEME USING THE DEPLOYMENT KNOWLEDGE

The trimming approach in the previous section considers all the sensor nodes as equals, but does not distinguish how likely two nodes can become neighbors. Generally speaking, in sensor networks, long distance peer-to-peer secure communication between sensor nodes is rare and unnecessary in many applications. That is, the public key authentication is mostly used by neighboring sensor nodes. Accordingly, if we know how likely two nodes are to become neighbors, we can achieve a more efficient memory usage by devoting relatively more memory to potential neighbors, as opposed to giving all sensors the same amount of memory. For instance, when considering how to maintain the Merkle forest in node A, if a node B is more likely to be A's neighbor, it is better to put B in a shorter Merkle tree; on the other hand, if there is a much slim chance for B to be A's neighbors, B can be put in a taller Merkle tree. In this way, the amortized energy consumption can be reduced.

To know how likely two sensors can be neighbors, certain degree of deployment knowledge must be known. In the next sub-sections, we will describe a group-based deployment model to model a general type of deployment knowledge. Then based on this deployment model, we trim the original Merkle tree into a set of Merkle sub-trees of different heights. We will show how to find the best height combination for each type of Merkle sub-trees, such that the communication overhead is minimized.

3.1 Deployment Knowledge Modeling

We assume that sensor nodes are static once they are deployed. We define *deployment point* as the point location where a sensor is to be deployed. This is not the location where this sensor finally resides. The sensor node can reside at points around this deployment point according to a certain probability distribution. As an example, let us consider the case where sensors are deployed from a helicopter. The deployment point is the location of the helicopter. We also define *resident point* as the point location where a sensor finally resides.

In practice, it is quite common that nodes are deployed in groups, i.e., a group of sensors are deployed at a single deployment point, and the probability distribution functions of the final resident points of all the sensors from the same group are the same.

In this work, we assume such a group-based deployment, and we model the deployment knowledge in the following (we call this model the *group-based deployment model*):

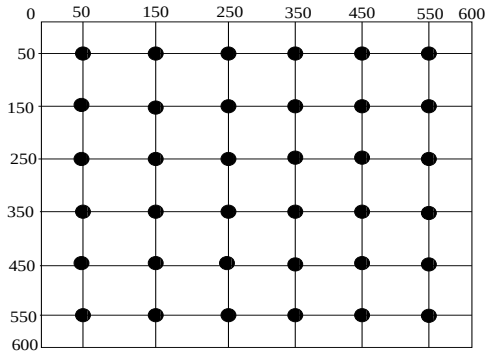


Figure 2: An Example of Group-based Deployment (each dot represents a deployment point).

1. N sensor nodes to be deployed are divided into n equal size groups so that each group, G_i, for $i = 1, \ldots, n$, is deployed from the deployment point with index i. To simplify the notion, we also use G_i to represent the corresponding deployment point, and let (x_i, y_i) represent its coordinates.

2. Locations of the deployment points are pre-determined prior to deployment. The deployment points can form any arbitrary pattern. For example, they can be arranged in a square grid pattern (see Figure 2), a hexagonal grid pattern, or other irregular patterns.

3. During deployment, the resident point of a node k in group G_i follows a probability distribution function $f_k^i(x, y \mid k \in G_i)$.

3.2 An Improved Authentication Scheme

For simplicity reason, in our following discussion, we assume that the deployment uses the grid pattern depicted in Figure 2. However, our general idea can be extended to other deployment patterns as well. The grid pattern illustrated here just serves to provide a convenient model for analysis, whose approach can also be applied to other group-based deployed patterns.

We classify each pair of deployment groups as horizontal/vertical neighbors, diagonal neighbors, and non-neighbors, based on their spatial relationship. To fully take advantage of Merkle trees, we let the number of sensors in each group be $S = 2^t$.

Our idea is that, if two nodes are from horizontal or vertical neighbor groups, they are more likely to be neighbors after the deployment than those from diagonal neighbor groups or from non-neighbor groups. Therefore, the height of the Merkle trees that these two nodes belong to should be different (preferably smaller) than nodes from other types of neighbor groups.

Here is how our improved scheme works: First, we build the same single Merkle tree as the one in the basic scheme. Then, for each node A, we use the following strategy to trim down the original Merkle tree into a set of Merkle sub-trees. Note, the strategy is node dependent; therefore, different nodes will end up having different Merkle forests.

- The sub-tree that corresponds to A's own group is trimmed down to Merkle sub-trees of height a ($a \leq t$).

- The sub-trees that correspond to A's horizontal or vertical neighbor groups are trimmed down to Merkle sub-trees of height b ($b \leq t$).

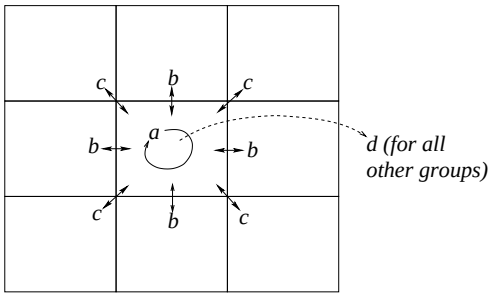


Figure 3: Height of Merkle Tree for nodes from different neighbor groups.

- The sub-trees that correspond to A's diagonal neighbor groups are trimmed down to Merkle sub-trees of height c ($c \leq t$).

- The original Merkle tree is trimmed down to Merkle sub-trees of height d. Note these sub-trees also contain the above three types of sub-trees. Such redundancy is inevitable when d is larger than t, the height of sub-trees for each group.

The above strategy is also depicted in Figure 3. An example of Merkle forest is depicted in Figure 4 (note that we only show a subset of all the groups in the example). In Figure 4, each solid black dot (both large and small ones) represents the root of a Merkle sub-tree. Each sensor nodes only needs to remember the hash value of these roots.

Memory Usage. Assume that the number of sensors in the network is N, and the number of sensors in each deployment group is S. Since the height of a Merkle tree for the nodes in the same group is a, and such a tree can accommodate 2^a nodes, the number of Merkle trees required for this group is $\lceil \frac{S}{2^a} \rceil$. This is also the number of hash values that need to be stored for this group. Similarly, we can compute the number of hash values for other neighbor groups. In total, the amount of hash values each node needs to carry in its memory is the following:

$$m = \lceil \frac{S}{2^a} \rceil + \lceil \frac{4S}{2^b} \rceil + \lceil \frac{4S}{2^c} \rceil + \lceil \frac{N}{2^d} \rceil \quad (1)$$

Finding the Values for a, b, c, and d. As we know, the larger the values of a, b, c, and d are, the more hash values need to be sent along with the public key. Therefore, if we have sufficient memory, we can choose $a = b = c = d = 0$ to minimize the communication overhead, i.e., we turn to the naive scheme described in Section 2. However, we do have a constraint on the memory usage; therefore we have an optimization problem: given the memory usage limit m_{max}, what are the optimal values for a, b, c, and d, such that the average communication overhead is minimized?

Before we can solve this question, we need to know how to compute the average communication overhead. We will derive the equations to do that, and then present how to solve the optimization problem.

3.3 Communication Overhead

Given any two neighbor nodes, we use w_0, w_1, w_2, and w_3 to represent the probability that these nodes are from the same group, from horizontal/vertical groups, from diagonal groups, and from

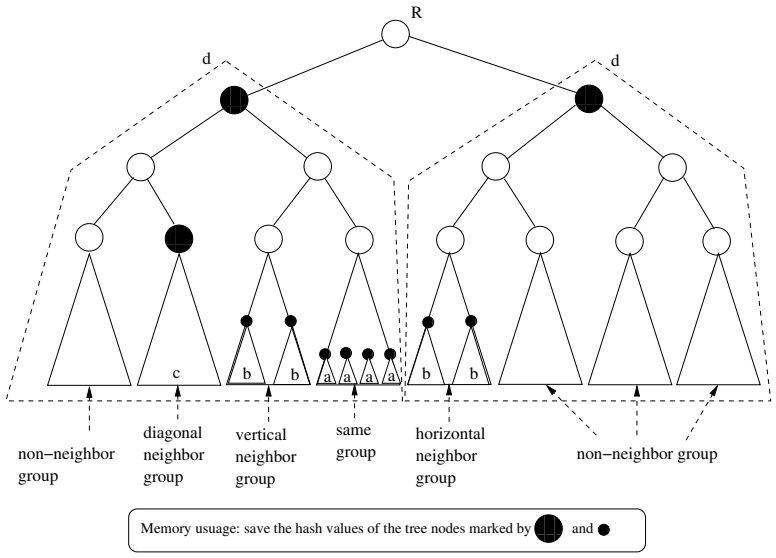
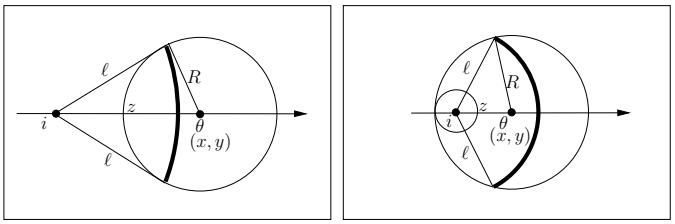


Figure 4: An example of Merkle Forrest.



(a) i is outside of the circle ($z > R$) (b) i is inside of the circle ($z < R$)

Figure 5: Probability of nodes residing within a circle.

non-neighbor groups, respectively. Therefore, the expected value for the communication cost can be computed as the following:

$$C = w_0 \cdot a + w_1 \cdot b + w_2 \cdot c + w_3 \cdot d.$$

In the following sections, we will show how to calculate the values for $w_0, \ldots, w_3$.

3.3.1 Computing the probability of being neighbors

We use z to represent the distance from point θ to the deployment point of group G_i. We define Ψ as the set of all deployment groups. We draw two circles. The first circle has a radius ℓ, and is centered at i, the deployment point of group G_i. We call this circle the i-circle. The second circle has a radius R, and is centered at $\theta = (x, y)$. We call this circle the θ-circle. When two circles intersect, we call the i-circle's arc within the θ-circle the L_{arc}, and we use $L_{arc}(\ell, z, R)$ to represent the length of the arc. We now consider an infinitesimal ring area $L_{arc}(\ell, z, R) \cdot d\ell$. The bold areas in Figure 5.a and 5.b show the infinitesimal ring areas.

Let $f(\ell \mid n_i \in G_i)$ represent the probability distribution function of the deployment for group G_i. Therefore, the probability that a node n_i from group $i \in \Psi$ with deployment point (x_i, y_i) resides within this small ring area is

$$f(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R) \cdot d\ell,$$

Using geometry knowledge, it is not difficult to derive the following equation for $L_{arc}(\ell, z, R)$:

$$L_{arc}(\ell, z, R) = 2\ell \cos^{-1}\left(\frac{\ell^2 + z^2 - R^2}{2\ell z}\right).$$

We define $g(z \mid n_i \in G_i)$ as the probability that the sensor node n_i from group i resides within the θ-circle, where z is the distance between θ and the deployment point of group G_i.

To calculate $g(z \mid n_i \in G_i)$, we integrate the probabilities over all the ring areas (for different ℓ) within the θ-circle. Therefore, when $z > R$ (as shown in Figure 5.a),

$$\begin{aligned} & g(z \mid n_i \in G_i) \\ =& \int_{z-R}^{z+R} f(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R)\, d\ell. \end{aligned}$$

When $z < R$ (as shown in Figure 5.b),

$$\begin{aligned} & g(z \mid n_i \in G_i) \\ =& \int_{0}^{R-z} \ell \cdot 2\pi f(\ell)\, d\ell \\ +& \int_{R-z}^{z+R} f(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R)\, d\ell. \end{aligned}$$

3.3.2 Compute $w_0, \ldots, w_3$

Assume n_i is a node from group i and n_j is a node from group j. Let $d_{i\theta}$ (resp. $d_{j\theta}$) represent the distance between θ and the deployment point of group i (resp. j). The following formula calculates the probability that n_j resides within the rectangle area $dx\, dy$ around point θ and n_i is a neighbor of n_j:

$$f(d_{j\theta} | n_j \in \text{ group } j) \cdot g(d_{i\theta} | n_i \in \text{ group } i) \cdot dx \cdot dy. \qquad (2)$$

We define Ψ_k be the set of pairs (u, v), where u and v represents indices of deployment groups, and

$$(u,v) \in \begin{cases} \Psi_0, & \text{when } u = v; \\ \Psi_1, & \text{when } u \text{ and } v \text{ are horizontal or vertical neighbor groups;} \\ \Psi_2, & \text{when } u \text{ and } v \text{ are diagonal neighbor groups;} \\ \Psi_3, & \text{when } u \text{ and } v \text{ are not neighbors.} \end{cases}$$

The value w_k is the average of the value in Eq. (2) throughout the entire deployment region, from $(0,0)$ to (X,Y), and for all the combinations of i and j that are in Ψ_k.

$$w_k = \int_{x=0}^{X}\int_{y=0}^{Y} \sum_{(i,j)\in\Psi_k} \Pr(n_j \in \text{ group } j)\Pr(n_i \in \text{ group } i) \cdot f(d_{j\theta}|n_j \in \text{ group } j)\, g(d_{i\theta}|n_i \in \text{ group } i)\, dx\, dy.$$

Since we assume that a sensor node is selected to be in each given group with an equal probability, we have

$$w_k = \frac{1}{n^2}\int_{x=0}^{X}\int_{y=0}^{Y} \sum_{(i,j)\in\Psi_k} f(d_{j\theta}|n_j \in \text{ group } j) \cdot g(d_{i\theta}|n_i \in \text{ group } i)\, dx\, dy, \quad (3)$$

where n is the number of deployment groups.

3.4 Find the Optimal Tree Heights

Recall that a, b, c, and d represent the height of the Merkle tree for the nodes in the same group, horizontal/vertical neighbor groups, diagonal neighbor groups, and non-neighbor groups, respectively. Given the memory constraints, we want to find out the optimal values of these parameters, such that the communication cost is minimized. The optimization problem can be formulated as the following:

PROBLEM 1. *(Find optimal a, b, c, and d) Let S be the number of sensors in each deployment group. Given that the maximum amount of memory (in the unit of the length of a one-way hash value) used for storing the roots of the Merkle trees is m_{max}, i.e.,*

$$\lceil\frac{S}{2^a}\rceil + \lceil\frac{4S}{2^b}\rceil + \lceil\frac{4S}{2^c}\rceil + \lceil\frac{n}{2^d}\rceil = m_{max},$$

find the values for a, b, c, and d, such that the communication overhead C of the following is minimized:

$$C = w_0 \cdot a + w_1 \cdot b + w_2 \cdot c + w_3 \cdot d.$$

The above optimization problem can be solved using brute-force techniques because we only have four variables (only three are free) and their values must be positive integers. The search space is $O((\log N)^3)$.

3.5 Incremental Deployment

It is possible that new sensors can be added to an existing sensor network. This is called *incremental deployment*. Existing sensors need to be able to authenticate the new sensors' public keys, and vice versa, new sensors should be able to authenticate the existing sensors' public key. For the new sensors to authenticate existing sensors, new sensors need to carry a Merkle forest formed by the existing sensors as well as the new sensors. If new sensors' deployment points are known apriori, we can use similar method to find the optimal configuration of the Merkle forest, such that the overall communication overhead is minimized.

Situations become more difficult for the existing sensors to authenticate new sensors' public keys. This is because the existing sensors' Merkle forests do not contain the new sensors. There are several solutions to deal with this problem. One solution is to prestore several Merkle trees into each sensor's memory, which contain the public keys that will be used in the future incremental deployment. However, this scheme limits the number of public keys that can be used in the future deployment. Another solution is to use the combination of the traditional certificate approach with our Merkle tree approach: we construct new Merkle trees for the new nodes, and let CA sign the root of the Merkle trees. Existing nodes need to get the new roots from any of the new nodes. They can use certificate verification to verify the authenticity of the root. Once the root is verified, all the public keys contained in this Merkle tree can be efficiently verified using our scheme. Because PKC is used only once for each Merkle tree, the amortized cost is still low.

4. PERFORMANCE EVALUATION

This section provides a detailed quantitative analysis evaluating the performance of our scheme. The metrics for the evaluation are the communication overhead, memory usage, computation costs, and energy consumptions.

In our experiments, the deployment area is a square plane of 800 meters by 800 meters. We use the square grid pattern for our deployment: namely, the plane is divided into 8×8 grids of size $100m \times 100m$; centers of these grids are chosen as deployment points. Figure 2 shows our deployment strategy. Similar experiments can be conducted for other deployment patterns. We use R to represent a sensor's transmission range. We set $R = 40$ in all of our experiments.

In our experiments, we model the sensor deployment distribution as a Gaussian distribution (also called Normal distribution). We assume that the deployment distribution for any node k in group G_i follows a two-dimensional Gaussian distribution, which is centered at the deployment point (x_i, y_i). Namely, the mean of the Gaussian distribution μ equals (x_i, y_i), and the pdf for node k in group G_i is the following [17]:

$$f_k^i(x, y \mid k \in G_i) = \frac{1}{2\pi\sigma^2} e^{-[(x-x_i)^2+(y-y_i)^2]/2\sigma^2}.$$

Therefore,

$$f(\ell \mid k \in G_i) = \frac{1}{2\pi\sigma^2} e^{-\ell^2/2\sigma^2},$$

where ℓ is the distance between (x, y) and (x_i, y_i).

4.1 Communication overhead vs. memory

The goal of this experiment is to study how memory usages affect communication overhead, and how much of communication costs can be saved using deployment knowledge.

For each experiment, we fix $\sigma = 50$ for the deployment distribution. We change the memory usage m from 1 to 200 (in terms of number of hash values), and then we compute the average communication overhead (also in terms of number of hash values). We conducted experiments for both the deployment knowledge-based scheme and the basic scheme. Recall that in the basic scheme, we also trim down the original Merkle tree to m sub-trees, but the trimming does not consider the deployment knowledge. The experiment results are depicted in Figure 6. The figure clearly shows that the communication overhead decreases with the increase of memory usages. More specifically, it shows that for network size 4096, the average communication overhead for the deployment knowledge-based scheme is less than two hash values. If we use 160-bit SHA1

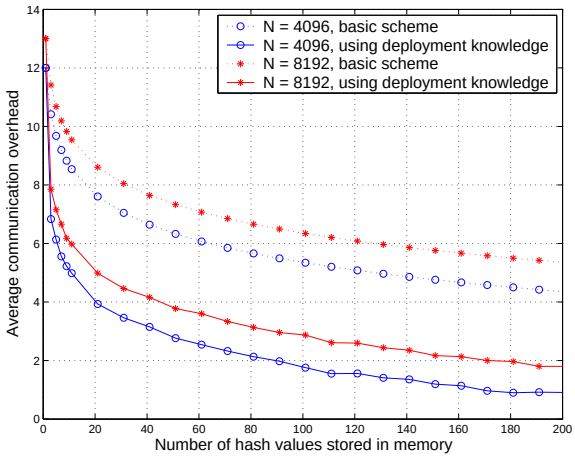


Figure 6: Communication overhead v.s. memory usages.

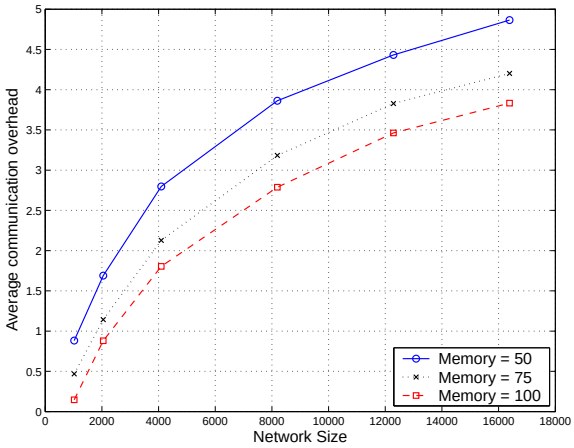


Figure 7: Communication overhead v.s. network size.

as our one-way hash function, the average overhead is less than 320 bits. The figure also shows that compared to the basic scheme, the deployment knowledge helps to save approximately 3.5 hash values in communication overhead.

When RSA public key algorithm is used, signatures on certificates can be at least 1024-bit if an 1024-bit private key is used. Therefore, compared to RSA, our public key authentication scheme not only saves computation energy (the goal of our scheme), but also saves communication costs. If we use ECC algorithm, the length of signatures can be reduced to 320-bit if a 160-bit [2] private key is used. In this case, our scheme has a greater communication overhead than ECC when the network size becomes large. However, the saving of our scheme on computation is so substantial that even with the extra communication overhead, our scheme still saves significant energy consumption. We will further analyze the energy consumption in Section 4.4.

4.2 Communication overhead vs. network size

The communication overhead of our scheme is proportional to the height of the tree, while the height of the tree is decided by the number of sensor nodes in the tree. Therefore, the size of a network will affect the performance of our scheme. On the other hand, the communication overhead for each single public key authentication is constant for traditional PKC algorithms. As a result, the larger the network, the less the energy savings of our scheme compared to traditional PKC algorithms. In this experiment, we show how the size of a network affect the communication overhead in our scheme. We only focus on our deployment knowledge-based scheme.

We conducted experiments for network sizes ranging from 1024 nodes to 16384 nodes, while setting the memory usage to 50, 75, and 100. Figure 7 depicts the results, which clearly show the trend of communication overhead with the increase of network size.

4.3 Impact of Deployment Knowledge

In this experiment, we evaluate how deployment knowledge affect the communication overhead. We study two issues of the deployment knowledge model: (1) the uncertainty of the deployment knowledge, and (2) the accuracy of the deployment knowledge model.

[2]NIST guidelines [2] points out that the security of a 160-bit ECC key is computationally equivalent to a 1024-bit RSA key.

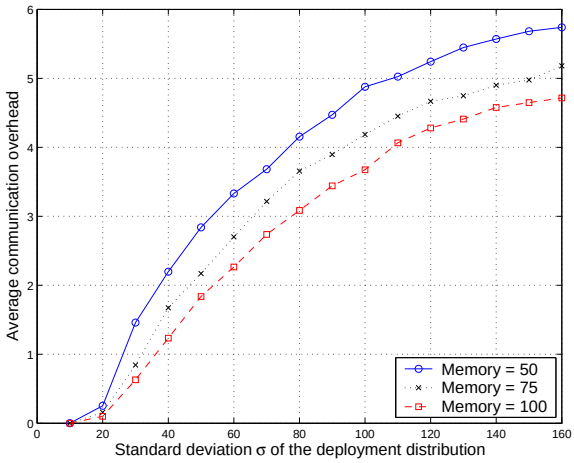


Figure 8: Communication overhead v.s. distribution (σ).

Uncertainty. Since we assume Gaussian Distribution in our evaluation, the uncertainty of the deployment knowledge is decided by the σ value of the distribution. The smaller the σ value, the less uncertainty we have about the deployment. In one extreme, when $\sigma = 0$, there is no uncertainty, and we know exactly where each sensor can land. On the other hand, when σ is large, sensors become widely spread, and neighborhood information becomes less useful.

In this experiment, we change the σ values of the distribution from 10 to 160, while we set the network size to 4096. Figure 8 depicts the results under different memory usage scenarios. It shows that the communication overhead increases with the increase of σ. Its reason is straightforward: when σ becomes large, the distribution becomes more even, thus the deployment knowledge becomes less useful. When σ increases to certain value, the performance of the deployment knowledge-based scheme can approach the performance of the basic scheme.

Modeling Accuracy. In practice, the deployment distribution in our model might not be the same as the final distribution. Our schemes achieves the minimal communication overhead based on the pre-deployment model; so the actual communication overhead will be different if the actual deployment distribution is different.

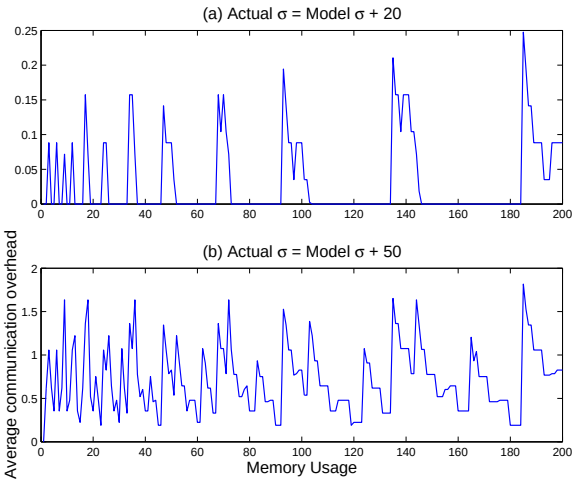


Figure 9: The effect of errors in modeling.

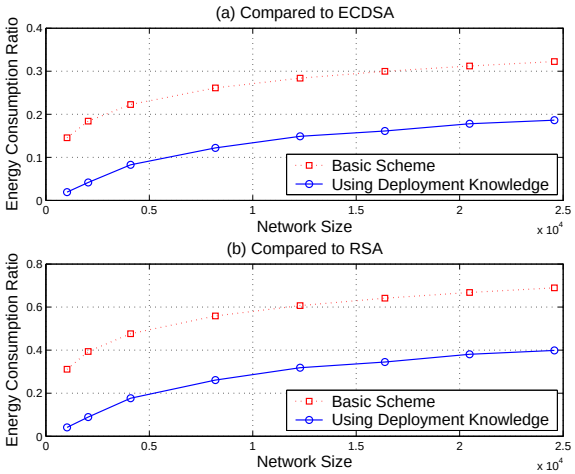


Figure 10: Comparison on energy consumption.

In this scheme, we study the impact of such inaccuracy.

We assume that the post-deployment distribution is still Gaussian, but the σ value of the distribution deviates from our model. In our experiment, we assume that the deployment model uses $\sigma_{model} = 50$, while the actual deployment distribution has $\sigma_{actual} = 50 + e$, where e is the error.

We can derive the optimal Merkle forest for the deployment model, and then compute the *actual communication overhead* when this Merkle forest is used in a distribution with σ_{actual}. We also compute the optimal communication overhead for σ_{actual}. We then plot the difference between the actual communication overhead and the optimal communication overhead. Figure 9 depicts the results for 8192 sensor nodes.

The results show that our scheme is not much sensitive to the small degree of modeling errors. For example, when the σ value deviates by 20, from 50 to 70, the difference of communication overhead between an accurate model and an inaccurate model is less than 0.25 (in the unit of hash-value length). However, when the modeling error is large, the difference becomes significantly large.

4.4 Comparison on Energy Consumption

In this experiment, we evaluate how much energy our scheme can save compared to the RSA algorithm and ECC algorithm. We get the performance data of RSA and ECC algorithms from [15] and the performance data of SHA1 from [13]. Both [15] and [13] conducted studies on ATmega128, a processor used for Crossbow motes platform [7]. However, since ATmega128 processor can run on both 8MHz and 16MHz modes, the evaluation in [13] chose 16MHz while the evaluation in [15] chose 8MHz. For the comparison purpose, we choose the 8MHz mode. We estimate the running time of SHA1 by multiplying the running time from [13] by two, with the assumption that it takes twice as much time to run an instruction on the 8MHz mode as that on the 16MHz mode. The performance of our scheme, RSA, and ECC on authenticating public keys are summarized in Table 1.

To compare energy consumption caused by computations with that caused by communications, we use an estimation made by Pottie and Kaiser, who pointed out that the energy consumed in transmitting a 1K-bit packet over 100m is approximately the same as performing 3 million instructions on a typical scenario [24]. Thus,

Table 1: Performance of authenticating public keys using various algorithms (k is the communication overhead in our scheme)

	Our Scheme (using SHA1)	RSA	ECC
Key or hash size (bit)	160	1024	160
Communication overhead (bit)	$160 \cdot k$	1024	320
Computation time (ms)	$7.2 \cdot k$	430	1620

with an 8MHz CPU, the energy spent on running CPU for 1 millisecond is equivalent to sending 2.67 bits of message.

Assume that the computation time for the traditional PKC-based public key authentication is T millisecond, and the communication overhead is C bits. Let k be the number of hash values that need to be sent to the verifier in our public key authentication scheme. The verifier needs to conduct k SHA1 hash evaluations. Therefore, the energy consumption ratio can be computed using the following formula:

$$ratio = \frac{160 \cdot k + 7.2 \cdot k \cdot 2.67}{C + T \cdot 2.67} \quad (4)$$

We have computed the ratio of our scheme to the RSA algorithm and ECC algorithm. For ECC, we assume that Elliptic Curve Digital Signature Algorithm (ECDSA) [1] is used. Our result is depicted in Figure 10. The amount of energy saving is substantial, especially for ECDSA algorithm. For example, our deployment knowledge-based scheme consumes only 14% of the energy used by ECDSA algorithm for a network of $10,000$ nodes while our basic scheme consumes 28%.

The saving over RSA algorithm is relatively less; this is because the signature verification of RSA is about four times as fast as that of ECDSA due to the small public key used in RSA. However, it is believed that ECC is more practical for sensor networks because RSA signature generation is quite slow: about 20 times slower than signature verification.

5. CONCLUSION AND FUTURE WORK

With the advance of hardware technology, Public Key Cryptography (PKC) will soon be available for sensor networks. However, compared to the secret key cryptography and one-way hash functions, PKC will still be much more expensive. To maximize the lifetime of batteries, the use of PKC in sensor networks must be limited and optimized.

This paper has shown that due to a unique property of sensor networks, public keys do not need to be authenticated in the same way as it is done in the Internet environment (i.e., using certificates); instead, public keys can be authenticated using one-way hash functions, which are much more efficient than signature verification on certificates. We have conducted extensive evaluation on our scheme. Our results show significant savings on power consumption. In our future work, we will focus on a variety of security protocols based on public key cryptography; we will investigate whether we can optimize PKC usages using the properties of sensor networks.

6. REFERENCES

[1] Ansi x9.62, elliptic curve key agreement and key transport protocols. American Bankers Association, 1999.

[2] NIST, special publication 800-57: Recommendation for key management. part 1: General guideline. Draft, January, 2003.

[3] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Proceedigns of CRYPTO, LNCS 2139*, pages 213–229, 2001.

[4] S.A. Camtepe and B. Yener. Combinatorial design of key distribution mechanisms for wireless sensor networks. In *Proceedings of 9th European Symposium On Research in Computer Security (ESORICS '04)*, 2004.

[5] D.W. Carman, P.S. Kruus, and B.J.Matt. Constrains and approaches for distributed sensor network security. Technical report, NAI Labs, 2000.

[6] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Research in Security and Privacy*, pages 197–213, 2003.

[7] Crossbow Technology Inc. Wireless sensor networks. http://www.xbow.com/. 2004.

[8] D. Medhi D. Huang, M. Mehta and L. Harn. Location-aware key management scheme for wireless sensor networks. In *2004 ACM Workshop on Security of Ad Hoc and Sensor Networks SASN03*, Washington, DC, USA, October 25 2004.

[9] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 42–51, Washington, DC, USA, October 27-31 2003.

[10] W. Du, J. Deng, S. Chen Y. S. Han, and P. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. In *Proceedings of the IEEE INFOCOM '04*, pages 586–597, Hongkong, China, March 27-31 2004.

[11] D. Eastlake and P. Jones. Us secure hash algorithm 1 (SHA1). IETF RFC 3174, September 2001.

[12] 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*, pages 41–47, November 2002.

[13] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu. Analyzing and modeling encryption overhead for sensor network nodes. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, San Diego, California, USA, September 19 2003.

[14] G. Gaubatz, J. Kaps, and B. Sunar. Public keys cryptography in sensor networks – revisited. In *The Proceedings of the 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS)*, 2004.

[15] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. August 11-13 2004.

[16] C. Karlof, N. Sastry, and D. Wagner. TinySec: Link layer encryption for tiny devices. In *ACM SenSys*, Baltimore, Maryland, USA, November 3-5 2004.

[17] A. Leon-Garcia. *Probability and Random Processes for Electrical Engineering*. Reading, MA: Addison-Wesley Publishing Company, Inc., second edition, 1994.

[18] D. Liu and P. Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, pages 263–276, February 2003.

[19] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 52–61, October 2003.

[20] D. Liu and P. Ning. Location-based pairwise key establishments for relatively static sensor networks. In *2003 ACM Workshop on Security of Ad Hoc and Sensor Networks SASN03*, Fairfax, VA, USA, October 31 2003.

[21] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *The First IEEE International Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, California, October 2004.

[22] R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Apr 1980.

[23] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks*, July 2001.

[24] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, pages 51–58, May 2000.

[25] R.L. Rivest, A. Shamir, and L.A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[26] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on embedded networked sensor system*, pages 188–200, Baltimore, MD, USA, November 3-5 2004.

[27] S. Zhu, S. Setia, and S. Jajodia. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 62–72, October 2003.