A Fast and Efficient Key Agreement Scheme for Wireless Sensor Networks

Mee Loong Yang*, Adnan al-Anbuky†, and William Liu‡
*School of Computing & Mathematical Sciences
Email: bobby.yang@aut.ac.nz
†School of Engineering
Email: aalanbuk@aut.ac.nz
‡School of Computing & Mathematical Sciences
Email: william.liu@aut.ac.nz
Auckland University of Technology, New Zealand

***Abstract*—The Blom's scheme for key agreement between pairs of nodes requires exchange of a small amount of bits, uses simple computations, and also authenticates each other. This makes it attractive for use in Wireless Sensor Networks but, in its original form, it has limitations because of the contending requirements for large pairwise keys and limited memory in the nodes. Our implementation of the Blom's scheme uses multiple keys, enabling it to derive large pairwise keys using the limited memory resources, while retaining all the desirable features of speed, compactness, and low energy usage. We implemented our scheme in a MICAz mote and present some experimental results on the memory, computation time, and energy requirements. We compared the performance with other public key cryptographic methods used in WSN. Our scheme, using 382 bytes of RAM, was able to compute 128-bits pairwise keys in times ranging from 34 ms to 1.9 s for networks with capture thresholds of 32 and about 2,000 nodes respectively.**

***Keywords*- Blom's scheme; ad hoc networks; security; wireless sensor networks; key pre-distribution.**

I. INTRODUCTION

A wireless sensor network deployed in open environments can be easily attacked. The radio communications can be eavesdropped, and an adversary can even inject malicious packets into the network. One important part in the chain of defence is to protect the communication channel between the nodes. This means encrypting the messages to protect confidentiality. In addition, the receiving node must be able to verify that the received messages are intact, fresh, and really originates from the claimed sender. Proven cryptographic techniques can be used to achieve these requirements. These techniques rely on the use of secret keys known only to the communicating parties. These keys must be distributed to the nodes in a secure manner.

One approach to key distribution requires the base station to generate all the keys which the nodes will use. Many early works take this approach using a global master key, such as in SPINS [1], and LEAP [2]. Obviously, if the master key is compromised, so is the whole network.

A more secure approach would use unique pairwise keys between nodes to limit the impact of any key compromise. However, in ad hoc networks, the pairwise relationships cannot be predetermined. In a network with N nodes, each node has $(N-1)$ pairwise keys with all its neighbours and a large amount of memory would be required to store them all. Even if this is possible, a single node compromised also affects the whole network. When deployed, nodes only need to have common keys with their immediate neighbours. Therefore, a probabilistic approach can be used. This was done in [3] where nodes are given subsets of keys from the global key pool. After deployment, pairs of nodes discover shared keys to establish secure links. Even without shared keys, pairs of nodes can attempt to use secured mutual intermediary nodes to establish a secure link. If one node becomes compromised, it only impacts on part of the network.

A different approach requires pairs of nodes to contact a trusted centre to provide their pairwise keys. Each node needs only to be provided with a pre-shared key with the trusted centre. This is the Key Distribution Centre (KDC) scheme used widely in protocols such as Kerberos [4]. In ad hoc mobile networks, this scheme is of limited use due to the requirement for the trusted centre to be reachable at all times.

Key pre-distribution schemes, on the other hand, pre-deploy nodes with keying material which they will use to derive pairwise keys with their neighbours. Such schemes commonly use public key cryptography (PKC). An example is the Diffie-Hellman (DH) protocol used widely in wired networks. For WSN, the Elliptic Curve DH (ECDH) is promising due to its less demand on resources compared to other PKC methods. Several implementations have been studied as in [5], [6], [7], [8]. These methods enable nodes to derive a common secret key by exchanging some information over the insecure channel. There is also the need for nodes to authenticate each other, usually using a certificate such as in the ECDH-ECDSA protocol implemented in [9], or using the Menezes-Qu-Vanstone (ECMQV) protocol [10], and that based on the ElGamal scheme [11]. We shall refer to these ECC methods later when comparing their performances with our scheme.

An interesting key pre-distribution scheme, which has implicit authentication, was proposed by Rolf Blom [12]. This scheme enables pairs of nodes to compute a common secret pairwise key after exchanging a small number of bits. The computation uses simple arithmetic operations and requires only a few steps. This makes it attractive for use in WSN. However, as shown later, in its basic form, it has limitations due to the contending requirements for small storage, large pairwise key sizes, and large number of nodes in the network. This paper describes our modifications to the scheme enabling it to derive large pairwise keys after exchanging a small number of bits. We also report the results of our implementation in a MICAz mote.

This paper is structured as follows. Section II presents the background and some related works. Section III describes our modifications to the Blom's scheme using multiple-keys. Next, in Section IV, we present the experimental results of our implementation in a MICAz mote. We discussed some of the security features and applications of our scheme in Section V. Then, we made some comparisons with other PKC methods in Section VI, and finally we gave our conclusion in Section VI.

II. Related work

A. Background: Blom's scheme

In this scheme, the base station generates a secret $(m \times m)$ symmetric matrix $\mathbf{S}$. Each node is assigned a unique $(m \times 1)$ column vector e.g., $\mathbf{V}_A$, and $\mathbf{V}_B$, for nodes A and B, respectively. These vectors are called the node's *public identifiers (IDs)* or *public vectors*. The base station then computes and stores in each node their private keys which are row vectors $K_x = \mathbf{V}'_x \cdot \mathbf{S}$. . These public IDs and private keys form the keying material for the node.

Any two nodes can derive a pairwise secret key between them. For example, between nodes *A* and *B*, the nodes exchange their public IDs and then compute a common key, K_{AB}.

$$\begin{aligned} \text{Node A: } K_{AB} &= (\mathbf{V}'_A \cdot \mathbf{S}) \cdot \mathbf{V}_B \\ \text{Node B: } K_{BA} &= (\mathbf{V}'_B \cdot \mathbf{S}) \cdot \mathbf{V}_A \\ K'_{BA} &= (\mathbf{V}'_B \cdot \mathbf{S} \cdot \mathbf{V}_A)' = \mathbf{V}'_A \cdot \mathbf{S}' \cdot \mathbf{V}_B \end{aligned}$$

Since $\mathbf{S}$ is symmetric, $K_{AB} = K'_{BA}$. An important feature is that success in deriving a common key authenticates the nodes to each other since this requires valid private keys provided by the base station.

In the key agreement process, the public IDs can be transmitted in clear text. The private keys must be kept secret. If an adversary captures the nodes, they may be able to obtain their keying material. If sufficient number of nodes are captured, these information can be used to derive the secret matrix $\mathbf{S}$ and hence break the whole scheme. For this to be possible, the number of nodes captured must be $\geq m$, and they must all have unique linearly independent public vectors. The system is said to be $(m-1)$ secure, i.e., a coalition of $(m-1)$ or less nodes cannot pool their keying material to derive the pairwise key of any other pair of nodes [13].

The column vectors of the Vandermonde matrix $\mathbf{V}$ given below is a suitable choice for public IDs since all s_k are distinct. The base station assigns to each node, one of the columns of $\mathbf{V}$. In practice, every node only need to be bootstrapped with the seed s_k from which to generate the public ID vector.

$$\mathbf{V} = \begin{bmatrix} 1 & \cdots & 1 & \cdots & 1 \\ s_1 & \cdots & s_k & \cdots & s_N \\ {s_1}^2 & \cdots & {s_k}^2 & \cdots & {s_N}^2 \\ \vdots & \ddots & \vdots & \ddots & \cdots \\ {s_1}^{m-1} & \cdots & {s_k}^{m-1} & \cdots & {s_N}^{m-1} \end{bmatrix}$$

Security parameters: If the network is $(m-1)-secure$, i.e., the matrix size is $(m \times m)$, and the size of each element of $\mathbf{S}$ is b -bits, then for a fully secure system,

$$\begin{aligned} \text{Max network size } Q_N &= (m-1) \text{ nodes} \\ \text{Public-ID seed size, } Q_v &= b \text{ -bits} \\ \text{Private-Key size, } Q_{ku} &= m \times b \text{ -bits} \\ \text{Pairwise key size, } Q_{kpair} &= b \text{ -bits} \end{aligned}$$

The pairwise key should be large, 64 -bits or larger, leading to large b. While the MICAz motes are able to work with 8, 16, 32 and 64 -bit data sizes, larger sizes for the vector elements would make the computations more complicated.

B. Related Works

The number of nodes in a fully secure network can be increased by using multiple key spaces. In [14], ω key spaces are generated and each node is given a sub-set of τ randomly chosen keys from ω. After deployment, nodes discover their common keys and use the Blom's scheme to form pairwise keys. The scheme uses a similar idea to the probabilistic scheme of Eschenaeur-Gligor [3] where nodes are given a random set of keys from a global key space. In these schemes the aim is to achieve full connectivity, but not necessarily complete connectivity like a full mesh. Another approach also uses Blom's scheme with multiple key spaces to improve resistance to the Sybil attack [15].

In [16], the scheme for a clustered topology is proposed. Here, the cluster-heads implement the Blom's scheme to derive pairwise keys among themselves. Non cluster-head nodes do not implement the Blom's scheme. Instead, they store a pre-computed secret key K_i for use with a cluster head. Prior to deployment, the base station computes the pairwise keys of this node with a certain number of associated cluster-heads. These are then combined into a secret key K_i and stored in the node, together with the identities (IDs) of the associated cluster-heads. When a node needs to establish a secure link with a physical cluster-head, it sends its own ID and the IDs of its associated cluster-heads. The physical cluster-head forwards the node's ID to the associated cluster-heads to compute the pairwise keys using Blom's scheme and thereby derives the secret key K_i. In this way, non-cluster head nodes store minimum keying material and do not need to perform any key computation computation. Instead, these are delegated to the cluster heads which carry the additional load of communicating with other cluster heads to derive the key with a non cluster-head node. The network size would still be limited to the $(m-1)$ nodes for a fully secure network. Since cluster-heads establish pairwise keys among themselves using the basic Blom's scheme, the key size and memory requirements, and network size would still be limited to the original scheme.

C. Our Main Contributions

We modified the Blom's scheme using multiple keys such that it is able to derive large pairwise keys of up to hundreds of bits using 16-bit data sizes. It requires very little RAM for the computation while retaining all the benefits of the basic scheme i.e., mutual authentication, few exchanged bits, simple computations, fast, and consumes little energy. This makes it especially suitable for dynamic B sensor networks where the nodes are highly mobile and key computation and authentication must be achieved quickly and cheaply. It is also scalable for fixed cluster based topologies.

III. Blom's scheme with Multiple-keys

In our modification to the scheme, the base station generates N secret symmetric $(m \times m)$ matrices $S_1, S_2, \cdots, S_N$, and one Vandermonde matrix $\mathbf{V}$, over the prime field $GF(p)$, where p is the largest prime $<b$ -bits. The number of bits, b is thus the data size used in the system.

Public IDs

Each node is given one unique set of vectors comprising N vectors of $\mathbf{V}$, called its *public ID vectors*. Since the elements of each vector is given by s_{ki} for $i = 0, \cdots, m-1$, an ID vector can be generated by anyone given s_k. Each node's public ID vectors can be simplified to the set of seeds $\{s_{k1}, s_{k2}, \cdots, s_{kN}\}$. We call this set the node's public *ID-tag*.

Private keys

For each node, the base station computes the set of private vectors or *private keys*, using all permutations of the secret matrices S_i and its public ID vectors. For node x, its private key set consist of N^2 separate $(1 \times m)$ row vectors given by,

$$K_{uxij} = \mathbf{V}'_{xi} \cdot S_j \quad \text{for } i, j = 1, \cdots, N$$

Pairwise Key Derivation

Consider two nodes *A* and *B* attempting to form pairwise key. Each has in their possessions, their public Id-tags and private keys:

$$\begin{aligned} \text{Node A:} \;\; & \{s_{A_i}\}, \{K_{uAj}\} \\ \text{Node B:} \;\; & \{s_{B_i}\}, \{K_{uBj}\} \\ \text{for} \;\; i = 1, \cdots, N & \text{ and } j = 1, \cdots, N^2 \end{aligned}$$

To derive their pairwise key they exchange their public ID-tags.

$$\begin{aligned} A \rightarrow B: & \quad \{s_A\} \\ A \leftarrow B: & \quad \{s_B\} \end{aligned}$$

Each node after receiving the other node's public ID-tag would generate that node's ID vectors, e.g., node B would be able to generate A's public vectors $V_{A_1}, V_{A_2}, \cdots, V_{A_N}$

Using all permutations of the public ID vectors and its own private keys, each node computes a set of secret numbers:

$$\begin{aligned} \text{Node A:} \quad & K_{uAj} \cdot \mathbf{V}_{B_i} \\ \text{Node B:} \quad & K_{uBj} \cdot \mathbf{V}_{A_i} \\ \text{for} \;\; i = 1, \cdots, N & \text{ and } j = 1, \cdots, N^2 \end{aligned}$$

Both parties would obtain a set of identical N^3 secret numbers, not necessarily arranged in the same order. Each number is of size b -bits. Using an agreed rule, sufficient numbers are chosen and concatenated together to form the pairwise key K_{ABpair} of the desired size. For example, a simple rule would be for each node to sort the numbers in descending order and concatenate the first 8 numbers to form a pairwise key of size $8 \times b$ -bits.

The key computation code is very simple and compact as shown in the pseudo code in Listing 1.

Listing 1. Pseudo code for pairwise key computation

```
generateKeyPair(){
  for (each public_ID-tag value) {
    generate public_vector;
    for (each private_key) {
      multiply with public vector;
      save result in SecretNumbers;
    }
  }
  sort SecretNumbers;
  select numbers from SecretNumbers;
  concatenate to form pairwise key;
}
```

Some parameters

Pairwise key size: The maximum pairwise key size, if all the secret numbers are used, is bN^3 -bits. With $N = 2$ and using 16 -bit data sizes, we have 128 -bits key sizes which is more than enough if it is used in a secure algorithm such AES.

Memory requirement: The private key requires the largest amount of memory. This is static data and can be assigned to program memory in the MICAz mote. The amount of RAM required include those for some counters, the pairwise key, some temporary data, and the N^3 secret numbers. In total, the RAM requirement is very small indeed as shown later.

Computation time: The main part of the computation involves modulo multiplication and addition of the m elements of the N public ID vectors and N^2 private key vectors. In total, there are mN^3 such operations.

IV. Experimental Results

We implemented the scheme in a MICAz [17] mote using TinyOS [18] for the case using 16 -bits data size. We also estimated the energy consumed for the key computation process. In our experiment, the code was kept to the bare minimum for key computation and turning on the LEDs at the start and end of process. Even though the modulo operations took a significant amount of time, no attempt was made to optimise it, and only the standard libraries were used for all operations.

The private keys were installed in the program code. The public ID-tag exchange was not implemented and was merely simulated by storing the public ID-tag of the simulated neighbour as a variable in the node.

When the program runs, it lights up an LED, computes the pairwise key, and lights up another LED on completion. The time taken for key computation was estimated by timing the 100 iterations of the computation. The power supply to the node was regulated at 3.1 V and the average current during computation was measured to be 8.7 mA.

Performance and analysis

The results for memory requirements, key computation time, and estimated energy consumed are shown in Table I.

Memory requirements: The MICAz mote has 4 kB RAM and 128 kB flash memory for program. In our implementation, we stored the private keys in program memory leaving more RAM for the variables. The ROM and RAM requirements were outputs from the TinyOS-2.1.1 compiler.

The private key vectors has mN^2 elements, each of 16 -bits. From the results, the ROM storage requirements in bytes was, as expected, a linear relationship as given below.

$$Q_O = 2.012\ mN^2 + 7010 \tag{1}$$

The number of variables in RAM was fixed except for the N^3 secret numbers. From the experimental results, the following relationship was obtained for the RAM storage requirements, in bytes:

$$Q_R = 8\ N^3 + 318 \tag{2}$$

The key computation process involves multiplying the m elements of N public ID vectors with the m elements of N^2 private key vectors, plus one sorting and selection operation. It was a linear relationship between the computation time and mN^3. From the results we obtained the following relationship for key computation time in ms,

$$t = 0.0514\ mN^3 + 24.60 \tag{3}$$

Computation energy: The average current drawn during the computation from the 3.1 V regulated power supply was measured to be about 8.7 mA. Using this, the estimated energy in mJ, used for computation was estimated as $3.1 \times 0.0087 \times t$, also shown in Table I.

Design example: The above results can be used for design purposes. For example, we have a network of 500 nodes and we wish to select the parameters and estimate the computation time and memory requirements. For a fully secure network, the number of nodes is $< \frac{m}{N}$. Trying with $N = 2$, a suitable choice is $m \geq 2 \times 500 = 1000$. Using a slightly larger value, $m = 1,024$, and $N = 2$ from (3) the key computation time would be 446 ms. The storage requirements, from (1) would be 15,252 bytes ROM, and from (2) gives 382 bytes RAM.

The results of an actual implementation in a MICAz mote was 15,282 bytes ROM, 384 bytes RAM, and the computation time was 479 ms.

V. Security features and applications

Brute force attacks: To attempt to guess the pairwise key or the private keys would be infeasible as these are large, at least 64 -bits, and hundreds of bits respectively.

Node capture

Nodes can be physically captured and sensitive information extracted unless tamper proof hardware mechanisms can be incorporated. This would increase the cost and probably not be viable except for critical situations. If such mechanisms are available, the scheme would be very attractive for highly mobile, ad hoc networks. For example, using a small $m = 24$ and $N = 2$ in (3), pairs of nodes can derive keys in 34.5 ms requiring about 0.93 mJ of energy.

Currently, motes do not have adequate tamper protection and an attacker with the appropriate skills and resources can easily obtain the memory contents from motes like the MICA2 [19], and TelosB [20].

The $(m-1)$ -secure property of the Blom's scheme still applies in our multiple-key case. If an attacker manages to obtain m sets of linearly independent public IDs and the associated private keys, it is possible to craft valid public

	Number of keys, N			
	1	2	3	4
matrix size: 64 × 64				
ROM (bytes)	6,888	7,678	8,312	9,206
RAM (bytes)	326	382	534	830
time (ms)	9	34	97	231
Energy (mJ)	0.24	0.92	2.62	16.23
matrix size: 128 × 128				
ROM (bytes)	7,016	8,192	9,466	11,260
RAM (bytes)	326	382	534	830
time (ms)	15	63	176	399
Energy (mJ)	0.40	1.70	4.75	10.76
matrix size: 256 × 256				
ROM (bytes)	7,274	9,210	11,772	15,358
RAM (bytes)	326	382	534	830
time (ms)	30	124	332	734
Energy (mJ)	0.81	3.34	8.95	19.8
matrix size: 1,024 × 1,024				
ROM (bytes)	8,812	15,354	25,596	*
RAM (bytes)	326	382	534	*
time (ms)	121	480	1,275	*
Energy (mJ)	3.26	12.95	34.39	
matrix size: 4,095 × 4,095				
ROM (bytes)	14,954	39,922	*	*
RAM (bytes)	326	382	*	*
time (ms)	486	1,906	*	*
Energy (mJ)	13.11	51.40	*	*

Table I
MEMORY REQUIREMENTS, COMPUTATION TIMES, AND ESTIMATED ENERGY FOR KEY COMPUTATION USING 16-BITS DATA SIZE. * THESE EXCEEDS THE ARRAY LIMIT

IDs and private keys for any other nodes. It would also be possible to derive the secret matrices and completely break the system. In our scheme, the number of captured nodes which can compromise the network, called the "capture threshold", is $Q_c = \frac{m}{N}$.

Depending on the application, we can implement the scheme with suitable levels of security for the following topologies,

- fully secure, ad hoc, mobile, full mesh topology
- fully secure, fixed, cluster topology,
- partially secure, ad hoc, mobile, full mesh topology

1. Fully secure, ad hoc, mobile, fully mesh topology: The scheme can be directly applied in this situation. All nodes are mobile and are able to form pairwise keys with every neighbour within range. The number of exposed nodes in the network is kept to below the capture threshold. Further, to prevent the captured keying material from being used to craft another node, the nodes have unique, non-intersecting sets of public ID vectors. This means there can be at most $\lfloor \frac{m-1}{N} \rfloor$ nodes in the network. This limits the network size to about 2,000 nodes for the case of $N = 2$, and $m = 4,095$ with the key computation time of about 1.9 s.

Smaller networks with highly mobile nodes such as in sports or combat situations would specially benefit from this scheme. For example, with about 30 nodes, using $m = 64$ and $N = 2$, nodes can derive a pairwise key in 34 ms using about 0.92 mJ of energy.

2. Fully secure, ad hoc, fixed cluster topology: If the network is organised as clusters and the nodes are fixed in position, we can implement the multiple-key Blom's scheme among the cluster head nodes, restricting their number to be less than the capture threshold. The leaf nodes also uses the scheme but with a difference in that after deployment, their public ID-tags are deleted once they have established a pairwise link with a cluster head, or within a certain time window. Without the ID-tags, the private keys cannot be used to compromise the system. Thereafter, the leaf node do not implement the scheme. For example, using $m = 4095$, $N = 2$, we can have up to about 2,000 cluster heads. If each cluster head has 10 nodes attached to it, the network size would be 2,500 nodes.

3. Partially secure, ad hoc, fully mesh topology: In some situations it is required to protect the network against casual or opportunistic attacks but not necessarily against determined and fully resourced adversaries such as rival organisations. In this case, since considerable resources in effort and time is required to capture Q_c nodes, extract the keying material, and solve the matrices, it may be permissible to exceed the capture threshold. Other security features such as node capture detection if implemented, would also help to support this approach. The network can then be much larger than the capture threshold. For example, if capturing 500 nodes and extracting the keying material is considered to be infeasible, we can have a thousand or more nodes in the network. The scheme can be directly implemented and have all the benefits of the ad hoc, mobile, fully meshed topology.

VI. DISCUSSIONS AND COMPARISONS

Comparision with PKC methods

The aim of our key agreement scheme is to derive large pairwise keys with authentication. Similar schemes which can achieve this are the PKC methods which have been successfully used in wired networks. Their application in WSN has been studied, such as TinyECC [21], EC-ElGamal [7], ECMQV/ECDCH [8], [6], etc. We will make some brief comparisons with our scheme in some performance metrics important in sensor networks, i.e., energy, key computation time, and memory requirements.

Exchange of keying material: Key agreement schemes requires the exchange of some keying material. The amount of bits exchanged impacts on the energy used for the radio. For comparison we will exclude the overheads such MAC addresses, protocol headers, etc.

The DH scheme requires the two parties to exchange their public keys from which to derive a common secret. To authenticate each other, the public keys need to be signed by a trusted authority. For ECC schemes, the basic components include a public key which is a point on the elliptic curve, its hash value, and the signature comprising two integers provided by the trusted authority. Using 160 -bits, an authenticated ECDH scheme would require exchange of at least 768 bits. Compared to our scheme requiring Nb -bits, this is larger by more than 10 times.

Key computation time and energy: In an optimised implementation of the ECDH scheme for WSN [6], the key computation took 710 ms, and used 17.04 mJ of energy in a MICAz mote running TinyOS. Another implementation using the EC-ElGamal scheme on a MICAz mote [7] reported 570 ms for key computation. These do not include signature verifications which would also require substantial amount of time and energy. For example, in [21], TinyECC was used to implement ECDH for key computation and ECDSA for signature verification in a MICAz mote. The results showed that with all optimizations enabled, the execution times were: ECDSA initialisation 3,393 ms, verification 2,436 ms, and for ECDH initialisation 1,839 ms, and key computation 2,117 ms. Hence, key computation and signature verification can take 4.5 seconds, after initialisation of about 5.2 seconds. Our key computation times depends on the choice of m and N as shown in Table I. In the largest case with $N = 2$ and $m = 4095$, the key computation took 1.9 seconds.

Memory requirements: The largest use of memory in our scheme is bmN^2 -bits for the private keys. This is static and can be stored in program memory. The code itself is very simple and compact requiring only a few hundred bytes of ROM storage. The total ROM memory for data code was less than 40 kB for the the largest values of $m = 4095$ and $N = 2$.

RAM storage was required only for variables such as the N^3 secret numbers, the pairwise key, and some counters. For the case of $N = 2$, only 382 bytes of RAM was required, as shown in Table I.

The PKC schemes require substantially more memory, especially RAM storage. In a MICAz mote implementing TinyECC [21], the memory requirements, with all optimisations enabled for ECDSA was 19,308 bytes ROM and 1,510 bytes RAM, for ECIES 20,758 bytes ROM and 1,774 bytes RAM, and for ECDH 16,018 bytes ROM and 1,774 bytes RAM. With all optimisations disabled, all the RAM sizes were only around 150 bytes but with consequently huge execution times, such as 31 seconds for ECDH key establishment! In [22] the code size for ECDSA as 56,600 bytes ROM and 1,700 bytes RAM.

VII. Conclusion

We presented our implementation of the modified Blom's scheme using multiple-keys which, while retaining the advantages of the basic scheme, improves it to make it very attractive for use in WSN. It is able achieve large pairwise key sizes, fast, and requires little energy and computational resources. We implemented our scheme in a MICAz mote and the results showed it be very advantageous compared other PKC schemes in terms of speed, energy, and RAM storage requirements. The network is fully secure if the number of compromised nodes do not exceed the capture threshold. The best choice was $N = 2$ keys, enabling pairwise key size of 128 bits requiring only 382 bytes of RAM. The ROM requirements are mainly for the node's private keys and its size depends on the capture threshold. In our case, the largest amount required was about 40 kB for a network with capture threshold of about 2,000 nodes. The key computation time increases as the capture threshold increases. This ranged from 34 ms for a capture threshold of 32 nodes, to 1.9 s for a capture threshold of 2,000 nodes using $N = 2$ keys.

References

[1] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "Spins: Security protocols for sensor networks," *Wireless Networks*, vol. 8, pp. 521–534, 2002.

[2] S. Zhu, S. Setia, and S. Jajodia, "Leap: Efficient security mechanisms for large-scale distributed sensor networks," *Proceedings of the 10th ACM conference on Computer and communications security*, 2003.

[3] 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*, pp. 41–47, 2002.

[4] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems." *In Proceedings of the Winter 1988 USENIX Conference. USENIX*, February 1988.

[5] M. Liu, W. Wei, and Z. Liu, "A secure key pre-distribution scheme for wireless sensor networks," *International Conference on Industrial Electronics and Applications, ICIEA.*, pp. 1762 –1768, May 2009.

[6] C. Lederer, R. Mader, M. Koschuch, J. Groschdl, A. Szekely, and S. Tillich, *Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks*. Springer Verlag, LNCS 5746, September 2009.

[7] O. Ugus, D. Westhoff, R. L. 0002, A. Shoufan, and S. A. Huss, "Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks," *2nd Workshop on Embedded Systems Security - WESS'2007, Salzburg, Austria.*, October 2007.

[8] J. Groschdl, A. Szekely, and S. Tillich, "The energy cost of cryptographic key establishment in wireless sensor networks," *Proc. The 2nd ACM Symposium on Information, Compter and Communication Security*, 2007.

[9] G. de Meulenaer, F. Gosset, F.-X. Standaert, and O. Pereira, "On the energy cost of communications and cryptography in wireless sensor networks," *IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, pp. 580–585, 10 2008.

[10] L. E. Law, A. J. Menezes, M. Qu, J. A. Solinas, and S. A. Vanstone, “An efficient protocol for authenticated key agreement,” *Designs, Codes and Cryptography*, vol. 28, no. 2, pp. 119–134, 2003.

[11] J. Zheng, J. Li, M. J. Lee, and M. Anshel, “A lightweight encryption and authentication scheme for wireless sensor networks,” *Int. J. Security and Networks*, vol. 1, no. 3/4, pp. 138–146, 2006.

[12] R. Blom, “An optimal class of symmetric key generation systems,” Linkopping University, Tech. Rep., 1984.

[13] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[14] W. Du, S. Y. Han, J. Deng, and P. K. Varshney, “A pairwise key pre-distribution scheme for wireless sensor networks,” *Proceedings of the conference on Computer and communications security*, October 2003.

[15] S.-J. Wang, Y.-R. Tsai, and J.-W. Chan, “A countermeasure against frequent attacks based on the blom-scheme in ad hoc sensor networks,” *International Symposium on Wireless Pervasive Computing*, 2007.

[16] N. Chen, J.-b. Yao, and G.-j. Wen, “An improved matrix key pre-distribution scheme for wireless sensor networks,” *International Conference on Embedded Software Systems*, p. 4045, 2008.

[17] *Memsic Corporation, MICAz Datasheet*. [Online]. Available: http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html, Retrieved: April 12, 2012

[18] P. Levis, *TinyOS programming*, 2006. [Online]. Available: http://csl.stanford.edu/ pal/pubs/tinyos-programming.pdf. Retrieved: April 12, 2012

[19] C. Hartung, J. Balasalle, and R. Han, “Node compromise in sensor networks: The need for secure systems,” Department of Computer Science, University of Colorado at Boulder, Tech. Rep., January 2005.

[20] A. Becher, Z. Benenson, and M. Dornseif, “Tampering with motes: Real-world physical attacks on wireless sensor networks,” *Proceedings of the Third international conference on Security in Pervasive Computing*, vol. 3934, pp. 104–118, 2006.

[21] A. Liu and P. Ning, “Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks,” *in Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, pp. 245–256, April 2008.

[22] H. Wang and Q. Li, “Efficient implementation of public key cryptosystems on mote sensors (short paper,” in *In International Conference on Information and Communication Security (ICICS), LNCS 4307*, 2006, pp. 519–528.