

Analysis of Proposed Key Establishment Protocols in Multi-Tiered Sensor Networks

Kalvinder Singh

Australia Development Lab, IBM and Griffith University

Email: kalsingh@au.ibm.com

and

Vallipuram Muthukumarasamy

School of Information and Communication Technology, Griffith University

Email: v.muthu@griffith.edu.au

Abstract—Wireless sensor networks provide solutions to a range of monitoring problems. However, they introduce a new set of problems mainly due to small memories, weak processors, limited energy and small packet size. Thus only a very few conventional protocols can readily be used in sensor networks. This paper closely examines the currently available key distributions protocols, their strengths and limitations. The performance of these protocols under different scenarios are thoroughly investigated using theoretical analysis and using a simulation study with TinyOS. First a number of single server protocols were proposed and their performance was analysed. Then we propose a new set of multi-server key distribution protocols, where base stations (or controller nodes) are untrusted. The proposed solutions replicate the authentication server such that a group of malicious and colluding servers cannot compromise security or disrupt service. A detailed comparison of the performance of the proposed protocols with that of other available protocols show the advantages of our protocols. We show that the proposed multiple server authentication protocols will only have $O(n)$ complexity, where n is the number of authentication servers. The protocols use information from the sensor nodes and the servers to generate a new key, and do not solely rely on the sensor nodes to generate good random numbers. The proposed protocols guarantee that the new key is fresh and that the communicating nodes use the same key.

Index Terms—wireless sensors, network security, key distribution, cryptography, performance

I. INTRODUCTION

Wireless sensors and actuators have the potential to significantly change the way people live and interact. As the sensors permeate the environment they can monitor objects, space and the interaction of objects within a space. Sensors can monitor a wide range of diverse phenomena by collecting information such as vibrations, temperature, sound, and light. Different sensors have different associated costs. For example, a sensor simply detecting light will have different costs to a sensor recording sound. However, less costly sensors can be used to detect a phenomenon before alerting the more costly sensors to start their monitoring. As the number of heterogeneous sensors increases, so will the amount of interactions between the sensors. In this paper, we propose a number of key establishment protocols that

can be successfully used in a Wireless Sensor Network (WSN).

There are many different types of sensor environments, ranging from large areas covered by sensors, to many sensors in a small area [1]. Different environments have a wide range of different characteristics. The protocols proposed in this paper are designed for Wireless Surveillance Sensor Networks (WSSNs), however, these protocols can also be applied to other environments that have similar security requirements found when using WSSNs.

A WSSN may not require encryption, but may require authentication. For instance, a sensor detecting the amount of light in a publically available room supplies no or little information to an adversary. However, the information the light sensor sends to another device should be authenticated before any actions are taken. TinySEC [2] is an example of link layer security, where encryption can be turned off, and Message Authentication Codes (MACs) are added to the packets to ensure the integrity of the messages.

Figure 1 is an example of a WSSN, consisting of cheap sensor nodes and more expensive cameras. The sensor nodes are motion detectors, and if they are triggered, they will notify the camera to start recording. Cameras sending continual data need to be connected to a high bandwidth and throughput network. For instance, Axis has wireless cameras that can connect to a 802.11g access point [3]. The Omnicast system can handle up to 50000 cameras, where the bottleneck is in the network [4]. Cameras are also designed to handle inputs and supply outputs to external sensors. SensEye [5] is an example of a multi-tier camera, where smaller cheaper cameras, with low bandwidth requirements, are used to notify more expensive cameras with higher bandwidth requirements of an event. Different networks can be used in surveillance systems, and each network has different requirements.

One of the security vulnerabilities we address in this paper is the use of small keys. For performance reasons, sensors use small keys, however, small keys give an attacker a greater opportunity to compromise a sensor node. Thus, small keys will require frequent refreshing. Old keys can be used to generate new keys. But, if the old key is compromised then the new key can easily be

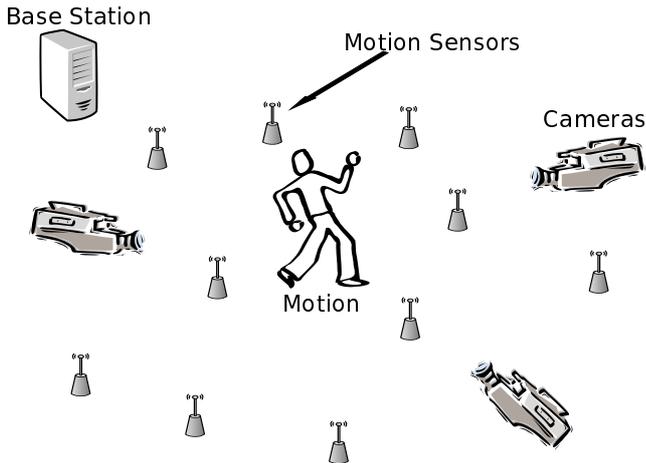


Figure 1. Motion Sensors Triggering a Camera in a WSN

compromised. A secure, efficient and scalable mechanism to freshen the keys between the sensors nodes is discussed in Section V-A.

When WSSNs are deployed in battlefields or developed to monitor homeland security, they have a likelihood of becoming the target of adversaries. In an open environment, an individual server or an intermediate sensor node may not be completely and permanently trustworthy. Several existing protocols [6], [7] found in traditional network protocols handle the shortcomings of untrusted servers. We, however, show that those existing protocols have an $O(n^2)$ complexity, and are therefore not suitable to a resource constrained environment.

In this paper, we propose five protocols to address the problems described above. Our proposed multiple server authentication protocols will have $O(n)$ complexity in time and space, and $O(n)$ messages, where n is the number of authentication servers. These protocols use information from the sensor nodes and the servers to generate a new key. The proposed protocols do not rely on the sensor nodes to generate cryptographically good pseudo-random numbers. We will show the sensor nodes can prove that the new key is fresh, and will demonstrate how key confirmation ensures that the nodes are guaranteed to be using the same key. We show a previously undiscovered attack against the KDC portion of the PIKE protocol. We also propose a defense against a well-known replay attack on the Boyd four-pass protocol [8]. Among the five proposed protocols presented in this paper, we have included two of our recently proposed [9] protocols as well. Although this paper is partly based on our previous work reported in [9], this paper has substantial new material including a number of new proposed protocols and in depth analysis.

Section II shows the notations used throughout this paper. Section III provides a background to WSNs and WSSNs, and the unique security challenges. Section IV describes the multiple server protocols found in traditional networks, and the problems with using those protocols in WSSNs. Section V examines single server protocols

found in both traditional and sensor networks, with the aim of converting one of them into a multiple server protocol. Two single server protocols are proposed and their performance is analysed in detail. Section VI proposes three multiple server protocols for WSSNs. Section VII provides a detailed analysis of the proposed multiple server protocols. Section VIII compares the proposed protocols in different sensor network environments. Finally, a conclusion is provided in Section IX.

II. NOTATION AND ASSUMPTIONS

Table I gives a list of notations, which are used when describing authentication and key establishment protocols in this paper.

TABLE I.
NOTATIONS

Notation	Description
A and B	The two nodes who wish to share a new session key.
S	A trusted server.
S_i	A server in a set of servers S_1, \dots, S_n , where n is the numbers of servers.
N_A	A nonce generated by A .
$\{M\}_K$	Encryption of message M with key K to provide confidentiality and integrity.
$[[M]]_K$	Encryption of message M with key K to provide confidentiality.
$[M]_K$	One-way transformation of message M with key K to provide integrity.
K_{AB}	The long-term key initially shared by A and B .
K'_{AB}	The value of the new session key.
K_{AS}, K_{BS}	Long-term keys initially shared by A and S , and by B and S for centralized authentication server.
K_{AS_i}, K_{BS_i}	Long-term keys initially shared by A and S_i , and by B and S_i , for each $i \in 1, \dots, n$.
X, Y	The result of the concatenation of data strings X and Y .
$A \rightarrow B : m$	Denotes that A sends a message m to B .
$A \rightarrow S_i : m$	Denotes that A sends a message m to each server.
$S_i \rightarrow A : m$	Denotes that each server sends a message m to A .
$X \oplus Y$	Exclusive-or operation with X and Y .

In protocols using a multiplicity of servers we also assume A and B do not trust any individual server. TinyOS is used as our development environment, and we used the following restrictions on the size of the data structures. The key size is 64 bits, the nonce size is 1 byte, the packet size is 29 bytes, and finally the location size is 2 bytes (which allows 64K of nodes in a sensor network).

III. BACKGROUND

We refer to a sensor network as a heterogeneous system combining small, smart, and cheap, sensing devices (sensors) with general-purpose computing elements. A sensor network consists of a potentially large number of sensors; there may also be a few control nodes, which may have more resources. The functions of the control nodes include: connecting the sensor network to an external

network; aggregating results before passing them on; controlling the sensor nodes; providing services not available to a resource constrained environment. Sensor network applications [10] include tracking bushfires, monitoring wildlife, conducting military surveillance, and monitoring public exposure to contaminants.

Some sensor nodes are resource constrained, such as the Mica mote [11]. The Mica motes contain a 4 MHz processor with 512 KB flash memory and 4 KB of data memory. A Mica mote also has a separate 512 KB flash memory unit accessed through a low-speed serial peripheral interface. The RF communication transfer rate is approximately 40 kbps. The maximum transmission range is approximately 100 meters in open space. Communication is the most expensive operation in sensor networks.

Other components in the sensor network may have more computation power and memory. Examples are the Stargate platform [11], the GNOME platform [12], the Medusa MK-2 platform [13], and the MANTIS platform [14]. These platforms may use other higher-level operating systems such as the Linux operating system. The platforms themselves may have additional communication mechanisms. For instance, the GNOME platform also has an Ethernet connection.

A. Key Establishment in Wireless Sensors

Security in sensor environments (where there are sensors with low resources) differs in many ways from that in other systems. Efficient cryptographic ciphers must still be used with care. Security protocols should use a minimal amount of RAM. Communication is extremely expensive; any increase in message size caused by security mechanisms comes at a significant cost. Energy is the most important resource, as each additional instruction or bit transmitted means the sensor node is a little closer to becoming nonfunctional. Nearly every aspect of sensor networks is designed with extreme power conservation.

There are many aspects to WSN security [15], [16]; ranging from data fusion security, location aware security, to the lower level security primitives such as cryptography, authentication and key establishment protocols. We will not cover all aspects of WSN security in this paper, instead only concentrate on some of the lower level primitives: authentication and key establishment protocols.

Several cryptography libraries using symmetric keys [17], [2] have been proposed. Recent work has shown that even asymmetric keys may be used in WSNs [18], [19]. Singh et al. [20] has proposed an efficient key establishment protocol using elliptic curves. However, they still consume considerably more resources than the symmetric counterparts.

Key establishment protocols are used to exchange and set up shared secrets between sensor nodes. Asymmetric cryptography is unsuitable for most sensor architectures because of the higher computational overhead, and energy and memory consumption. When using symmetric

keys, we can classify the key establishment protocols in WSNs into three main categories: Pair-wise schemes; Random key predistribution schemes; Key Distribution Center (KDC) schemes.

The simplest is the full pair-wise scheme [21], where each node in a network of total of n nodes shares a unique pairwise key with every other node in the network. The memory overhead for every sensor node is $(n - 1)$ cryptographic keys. Other pair-wise schemes [22], [23] also have $O(n)$ memory cost. In a pair-wise scheme, the sensor network is not compromised even if a fraction of the sensors are compromised.

Random key predistribution schemes are the second category [21]. This is a major class of key establishment protocols for sensor networks. They rely on the fact that a random graph is connected with high probability if the average degree of its nodes is above a threshold. After the connected secure network is formed, the protected links can be further used for agreeing on new keys, called *path-keys*. One of the main problems with random key predistribution schemes is that if a certain number of sensor nodes become compromised, then the entire sensor network can be compromised.

The third category is the KDC scheme. If two entities sharing no previous secret want to communicate securely with each other, they can receive assistance from a third party. In WSNs the two entities are typically resource-constrained sensor nodes, and the third party is a resource-heavy base station. However, in a multi-tiered environment, such as a WSSN, the third party may be a resource heavy camera. Typically, the base station provides an authentication service that distributes a secure session key to the sensor nodes. The level of security of a typical key distribution protocol depends on the assumption that the third party is trustworthy [8].

KDC schemes use the least amount of memory compared with the other two categories, and has an extra advantage of providing authentication for the sensor nodes. Examples of KDCs in WSNs were first proposed in SPINS [17]. However, the SPINS protocol may not be suited for every WSN topology. For instance, it does not easily scale to a large WSN, since the non-uniform communication will focus the load onto the KDC. This may cause the battery life of the network nodes to diminish considerably. However, a KDC mechanism is suitable for a surveillance sensor environment.

Hybrid schemes can also be created by combining different key establishment categories. The PIKE scheme [24] is such a scheme, it combines a pair-wise scheme with the KDC scheme, where one or more sensor nodes act as a trusted intermediary to facilitate the key establishment. The scheme was developed to limit the amount of memory used by the pair-wise and random key predistribution schemes, and also to limit the communication load because of the KDC schemes. However, the difficulties in using a sensor node as the trusted third party are: the trusted intermediary can easily become compromised; key sizes in sensors nodes are not large; sensor networks

may only accept authenticated messages, so may not have access to an encryption algorithm.

It should be noted that none of the above sensor key establishment schemes can handle authentication when the third party is compromised. Also, if the KDC is a sensor or controller node, then there is a higher likelihood that it can be compromised.

B. Security for the KDC

Several researchers have addressed security of the controller nodes and/or the base station. SIA [25] addresses the issue of compromised nodes by using statistical techniques and interactive proofs, ensuring the aggregated result reported by the base station is a good approximation to the true value, even if a small number of sensor nodes and the aggregation node may have been compromised. However, the communication overhead between sensor nodes and the base station is high. Other works have shown that some of the statistical methods used are not resilient to a group of malicious sensor nodes, and the end user should be aware of which statistical methods are easily cheated [26]. Another way to protect results is to use a witness node mechanism [27].

A different approach is to protect the base station location. Routing mechanisms to protect the location and disguise the identity of the base station have been proposed [28]. Hop-by-hop re-encryption of each packet's header and data fields is designed to change the presentation of a packet so that it cannot be used to trace the direction toward or away from the base station. Uniform rate control is advised so that traffic volume nearer the base station is undifferentiated from traffic farther from the base station. Time decorrelation between packet arrivals and departures further increases the difficulty of tracing packets.

However, ensuring that the authentication services are not hindered by a compromised or broken controller node or base station presents different challenges. A simple approach is to replicate the authentication services of the server so that any one of several servers can perform authentication. However, this approach reduces the level of security; if one server is compromised, security for every replicated server is compromised.

C. Limitations and Concerns

When WSNs are deployed in battlefields or developed to monitor national security, they have a likelihood of becoming the target of adversaries. In an open environment, an individual server or an intermediate sensor node may not be completely and permanently trustworthy. To make a key distribution protocol work in an environment where sensor nodes do not trust an individual base station, an authentication scheme, which can be used with limited resources and can reduce the requirement for trusting servers, needs to be found.

For performance reasons, sensors use small keys. However, small keys give an attacker a greater opportunity to

compromise a sensor node. Thus, small keys will require frequent refreshing. Old keys can be used to generate new keys. But, if the old key is compromised than the new key can easily be compromised. Many existing sensor protocols concentrate on the initial key distribution but do not have a secure mechanism available to update the keys. Keys in sensor networks are usually 64 bits in size, and they may become easily compromised. Sensors that are short-lived may not require that cryptographic keys be updated, however any sensors that exist for an extended amount of time will require updating of keys.

IV. PROBLEMS IN USING MULTIPLE SERVERS

Boyd and Mathuria have produced a survey of key establishment protocols using multiple servers [29] in traditional networks. In their survey, two multiple server protocols were listed: Gong's multiple server protocol [6], and the Chen–Gollmann–Mitchell protocol [7]. However, this survey did not take into account the unique nature of a sensor environment. The main goals of using multiple servers in a sensor network are:

- even if one or more servers become unavailable, it may be possible for the sensor nodes to establish a session key.
- even if one or more servers are untrustworthy, the sensor nodes may still be able to establish a good key.

Protocol 1 Gong's simplified multi-server protocol

M1	$A \rightarrow B :$	$A, B, N_A, \{A, B, x_1, cc(x)\}_{K_{A1}}, \dots, \{A, B, x_n, cc(x)\}_{K_{An}}$
M2	$B \rightarrow S_i :$	$A, B, N_A, N_B, \{A, B, x_i, cc(x)\}_{K_{Ai}}, \{B, A, y_i, cc(y)\}_{K_{Bi}}$
M3	$S_i \rightarrow B :$	$\{B, N_A, y_i, cc_i(y)\}_{K_{Ai}}, \{A, N_B, x_i, cc_i(x)\}_{K_{Bi}}$
M4	$B \rightarrow A :$	$\{B, N_A, y_1, cc_1(y)\}_{K_{A1}}, \dots, \{B, N_A, y_n, cc_n(y)\}_{K_{An}}, \{N_A\}_{K_{AB}}, N_B$
M5	$A \rightarrow B :$	$\{N_B\}_{K_{AB}}$

A simplified version of Gong's original multiple server protocol is described in [29]. We describe this version as shown in *Protocol 1*. One of the main features of this protocol is that the nodes, A and B , choose the keying material while the n servers, S_1, S_2, \dots, S_n , act as key translation centers that allow keying material from one node to be made available to the other. Initially A shares a long-term key K_{Ai} with each server S_i , and similarly B shares K_{Bi} with S_i . Node A has split the key x into x_1, x_2, \dots, x_n and node B has split the key y into y_1, y_2, \dots, y_n . The session key is defined as $K_{AB} = h(x, y)$ where h is a one-way function. The protocol sends a total of $2n + 3$ messages.

To prevent compromised servers from disrupting the protocol, A and B form a cross-checksum for all the shares. The cross-checksum for x is shown in Equation (1).

$$cc(x) = (h(x_1), h(x_2), \dots, h(x_n)) \quad (1)$$

The $cc_i(x)$ (should be equal to $cc(x)$) and $cc_i(y)$ (should be equal to $cc(y)$) are the cross-checksums returned by server S_i . The node will give a credit point to the servers if their cross-checksum values are the same as the values obtained from the majority of servers. When all the checks are complete, B retains the value x_j with the most credit points.

The major problem with this protocol is the size of the messages. The message sizes of $M1$ and $M4$ in the Gong multi-server protocol are of $O(n^2)$. Message $M5$ is $O(1)$, while $M2$ and $M3$ are of $O(n)$. A message size of $O(n^2)$ is not desirable in a sensor network. Another problem is that the size of the output of the one-way function will have to be reasonably large (otherwise a malicious server can quickly calculate the possible values for x and y). So for small values of n , the message sizes themselves will be very large for a sensor network.

The second multiple server protocol we consider is the Chen et al. multiple server protocol as shown in Protocol 2. One of the main features of this protocol is that the servers, rather than the sensor nodes, choose the keying material. Both nodes employ a cross-checksum to decide which servers have given valid inputs. The protocol sends a total of $2n + 4$ messages.

Protocol 2 Chen-Gollmann-Mitchell multi-server protocol

$M1$	$A \rightarrow B :$	A, B, N_A
$M2$	$B \rightarrow S_i :$	A, B, N_A, N_B
$M3$	$S_i \rightarrow B :$	$\{B, N_A, K_i\}_{K_{Ai}}, \{A, N_B, K_i\}_{K_{Bi}}$
$M4$	$B \rightarrow A :$	$\{B, N_A, K_1, \dots\}_{K_{A1}}, \dots,$ $\{B, N_A, K_n\}_{K_{An}}, cc_B(1), \dots,$ $cc_B(n)$
$M5$	$A \rightarrow B :$	$cc_A(1), \dots, cc_A(n),$ $\{B, N_B, N'_A\}_{K'_{AB}}$
$M6$	$B \rightarrow A :$	$\{A, N'_A, N_B\}_{K'_{AB}}$

The cross-checksum used in this protocol is different from the one used in the Gong multi-server protocol. The sensor node B calculates the $cc_B(i)$ as shown in Equation(2).

$$cc_B(i) = \{h(K_1), h(K_2), \dots, h(K_n)\}_{K_i}, \forall i \in (1, \dots, n) \tag{2}$$

To prevent A or B imposing the session key, the choice of $h()$ is limited; for example, it cannot be an exclusive-or-operation. If B doesn't receive any message from server S_j then $cc_B(j)$ is an error message, and $h(K_j)$ is replaced by an error message in the calculation of the other $cc_B(i)$ values. When A receives the checksums, A will first decrypt the values and compares the values with its own calculations of the cross-checksums. The valid K_i secrets are retained for the majority of i values and others are discarded. The session key K_{AB} is defined to be the hash of all the good K_i values concatenated, as shown in Equation (3).

$$K_{AB} = h(K_j, \dots, K_m) \tag{3}$$

The messages $M4$ and $M5$ in the Chen et al. multi-server protocol have a computational complexity of $O(n^2)$. While messages $M2$ and $M3$ have $O(n)$, messages $M1$ and $M6$ have $O(1)$ computational complexity.

This protocol encounters similar problems as the Gong multi-server protocol with regard to the size of the messages. Once again, several messages are of $O(n^2)$ in size. With the cross-checksums containing the outputs of a one-way function where the inputs are key values, once again the size of the output will need to be large. The cross-checksums in this protocol are encrypted instead of only requiring a hash algorithm. However, the Chen et al. multi-server protocol is considerably more efficient with regard to the size of the messages.

Another aspect of the multiple server protocols is the creation of the new key K_{AB} . The nodes A and B retrieve the new key by using a secret sharing mechanism such as the one defined in [30]. Secret sharing is a mechanism allowing the owner of a secret to distribute shares amongst a group. Individual shares or a small number of shares are no help in recovering the secret. The n shares are distributed, such that any set of t (or more) shares is sufficient to obtain the secret. The most well-known threshold scheme uses polynomial interpolation.

When polynomial interpolation is used in cryptographic applications the field is typically \mathbb{Z}_p , the field of integers modulo p , for some prime p . To share a secret $s \in \mathbb{Z}_p$ in the (t, n) threshold scheme, the dealer generates a polynomial of degree $t - 1$:

$$f(x) = a_0 + a_1x + \dots + a_{t-1}x^{t-1} \tag{4}$$

The coefficients are randomly chosen in \mathbb{Z}_p except for $a_0 = s$. The shares are values of $f(x)$ with $1 \leq x \leq n$. If t shares are known s can be recovered. For example, if $f(1), f(2), \dots, f(t)$ are known then:

$$s = \sum_{i=1}^t f(i) \prod_{i < j \leq t} \frac{j}{j-i} \tag{5}$$

Given any t points on the polynomial (excluding the value of 0), the value for $f(0)$ can be obtained.

The time complexity to compute n shares is $O(nt)$. The time complexity to recover a_0 is $O(t \log_2 t)$ [6]. Having such a scheme or similar scheme in a sensor node will consume significant amount of resources in an already resource constrained environment.

The existing multiple server protocols are therefore not suitable for a sensor network environment. An ideal solution with the desired characteristics requires innovative multiple server protocols.

V. SINGLE SERVER PROTOCOLS

We investigated the available single server protocols with the intension of exploring the possibility of extending one of the protocols into a suitable multiple server protocol. We also developed two single server protocols and compared their performance, both analytically and via simulation, with the existing protocols. Of particular

interest are the protocols using shared key cryptography. Boyd and Mathuria have also surveyed the protocols for authentication and key establishment [29] in traditional networks. However, the protocols have not been extensively analyzed for WSN environments.

Boyd and Mathuria have listed a total of 22 server-based key establishment protocols [29]. For our analysis we have reduced the number of significant protocols to a manageable size of seven. Keeping sensor environments in mind, we have used the following steps to filter out the desired protocols. Protocols requiring sensor nodes that store old messages to prevent replay attacks have been removed. If an optimized version of the protocol exists, it will be looked at in preference to the older protocol. The requirement of a flexible protocol for most situations helped us rule out the protocols relying on timestamps, as not all sensor environments are guaranteed to have secure time synchronization. Another requirement is to minimize the amount of communication, so we removed any protocols requiring five or more messages, leaving us with protocols sending only four messages, as shown in Table II.

TABLE II.
COMPARISON OF FILTERED TRADITIONAL NETWORK SINGLE
SERVER PROTOCOLS

Protocol	Properties for the Key		
	Control	Freshness	Confirm
Bauer–Berson–Feiertag [31]	S	A+B	No
Otway–Rees [32]	S	A+B	No
Yahalom [33]	S	A+B	B
Bellare–Rogaway [34]	S	A+B	No
AN Otway–Rees [35]	S	A+B	No
11770–2 Mech. 13 [36]	B	A+B	No
Boyd four–pass [8]	S/A/B	A+B	A+B

From the remaining seven protocols, the protocol with the most desired properties was the Boyd four–pass protocol [8]. One of the most important properties is the key confirmation property: if key confirmation is not considered important in a particular environment, then another protocol such as Bellare–Rogaway can be considered.

The Boyd four–pass protocol provides key authentication, key freshness and key confirmation in four messages, as shown in *Protocol 3*. The nodes A and B exclusively share a secret (K_{AS} and K_{BS} respectively) with the trusted authentication server S . By executing the protocol sensor nodes A and B intend to establish a session key K_{AB} .

Protocol 3 Boyd key agreement protocol

$M1$	$A \rightarrow S :$	A, B, N_A
$M2$	$S \rightarrow B :$	$\{A, B, K_S\}_{K_{AS}}, \{A, B, K_S\}_{K_{BS}}, N_A$
$M3$	$B \rightarrow A :$	$\{A, B, K_S\}_{K_{AS}}, [N_A]_{K_{AB}}, N_B$
$M4$	$A \rightarrow B :$	$[N_B]_{K_{AB}}$

An attractive feature of this protocol is that K_{AB} is

generated using information from A , B and S , as shown in Equation (6). The nonces are used to guarantee the key is fresh. The nonces N_A and N_B can either be random numbers or a counter, although K_S should be a good random number. The last two messages supply the key confirmation functionality for A and B . Also, the server should remove K_S from memory. If the server itself ever becomes compromised, then it will not compromise any sensor nodes.

$$K_{AB} = [N_A, N_B]_{K_S} \quad (6)$$

The requirement for $M2$ and $M3$ to have the location of A and B encrypted is not essential. Thus, we have proposed a modified version of the Boyd four–pass protocol. The *Modified Protocol 1* is the modified version without the encryption of the locations, however the locations are still used to create the MAC values.

Modified Protocol 1 Modified Boyd key agreement protocol

$M1$	$A \rightarrow S :$	A, B, N_A
$M2$	$S \rightarrow B :$	$[[K_S]]_{K_{AS}}, [A, B, K_S], [[K_S]]_{K_{BS}}, [A, B, K_S]_{K_{BS}}, N_A, A$
$M3$	$B \rightarrow A :$	$[[K_S]]_{K_{AS}}, [A, B, K_S]_{K_{AS}}, [N_A]_{K_{AB}}, N_B$
$M4$	$A \rightarrow B :$	$[N_B]_{K_{AB}}$

The security of the protocol is slightly weakened because the location names are no longer verified from both decrypting the message and the integrity checks of the MACs. When B and A receive their messages, the location names B and A can only be verified from the MACs. However, conventional security protocols err on the side of caution [2]. Most algorithms producing MACs are good enough, because the probability that the location names are not A and B is extremely low. The benefit to performance is considered to be worthwhile, at the cost of a minimal decrease in security. In the next sub–section, we investigate the scalability of our *Modified Protocol 1*.

A. Scalability Improvements when Updating Keys

The *Modified Protocol 1* will only need to be run once between A and B . The sensor nodes can cache K_S and instead of contacting the server again, they can then use a different protocol to establish a new key. Upon further investigation, we discovered that we can exploit the use of Bellare–Rogaway MAP1 protocol [37], a provably secure entity authentication protocol, to produce a new session key.

The MAP1 protocol, as shown in *Protocol 4*, provides mutual authentication between A and B . If K_{AB} is calculated using Equation (6), then this protocol becomes a key establishment protocol. The new K_{AB} key is guaranteed to be fresh, since N_A and N_B are used to create the new key. Key confirmation for both sensor nodes is another feature of this protocol. By creating a new K_{AB} we have

Protocol 4 Bellare–Rogaway MAP1 protocol

M1 $A \rightarrow B : A, N_A$
M2 $B \rightarrow A : N_B, [B, A, N_A, N_B]_{K_{AB}}$
M3 $A \rightarrow B : [A, N_B]_{K_{AB}}$

transformed an entity authentication protocol into a key establishment protocol.

One of the properties of the MAP1 protocol is that encryption is not used to establish the key. There are situations where wireless sensor environments do not need to support encryption, and may only need integrity checking. However, most key establishment protocols, where there is a trusted server involved, require some form of encryption. The reason for the integrity checking is that Bellare–Rogaway MAP1 protocol has key confirmation for both the sensor nodes. Key confirmation functionality can be removed if we remove message M3, and remove the MAC from M2.

B. Removing Encryption Requirements from the Boyd Protocol

Janson and Tsudik developed an authenticated key distribution that did not require traditional encryption when establishing a new session key [38]. We extend their technique to remove the need for encryption in our proposed protocol. In our *Proposed Protocol 1* [9], the following constructs are used:

$$\begin{aligned} AUTH_A &= [A, B, K_S]_{K_{AS}} \\ MASK_A &= [[AUTH_A]]_{K_{AS}} \\ AUTH_B &= [A, B, K_S]_{K_{BS}} \\ MASK_B &= [[AUTH_B]]_{K_{BS}} \end{aligned}$$

The MASK can either be created from an encryption algorithm or from a MAC. In cases like CBC–MAC, where the algorithm uses an underlying encryption algorithm, it may be more efficient to use encryption. In other cases where there is no encryption algorithm available (for instance there is only HMAC–MD5, or hardware support) then the MASK can be created by using the MAC. The size of MASK should be equal to or greater than the size of the keys. By default TinySEC has a 32 bit MAC; however, the CBC–MAC can create a variable size MAC. We have also split message M2, allowing us to decrease the size of message M3, thus minimizing the amount of data sent by sensor node B. However, if desired, messages M2 and M2' can be recombined into a single message, and then message M3 will need to increase in size again. Also, if key confirmation functionality is not required then messages M3 and M4 can be removed.

Proposed Protocol 1 No Encryption Key Agreement Protocol

M1 $A \rightarrow S : A, B, N_A$
M2 $S \rightarrow B : A, N_A, AUTH_B, MASK_B \oplus K_S$
M2' $S \rightarrow A : AUTH_A, MASK_A \oplus K_S$
M3 $B \rightarrow A : [N_A]_{K_{AB}}, N_B$
M4 $A \rightarrow B : [N_B]_{K_{AB}}$

C. Security Analysis of Proposed Protocol 1

In *Proposed Protocol 1* there are two messages that contain the MASK; the message M2 – server S sending to node B, and the message M2' – server S sending to node A. Neither A nor B ever send out the MASK. A possible attack on our protocol is for an adversary to try and obtain the MASK value by interrogating S. If an adversary pretends to be A, it does not matter what locations and nonce gets passed to S, because S should produce a new K_S , and therefore a new MASK and a new $MASK \oplus K_S$.

As shown in Equation (7), if two exclusive–ors produce the same value, and the keys are different, then the MASK will have to be different.

$$MASK \oplus K_S = MASK' \oplus K'_S \quad (7)$$

If the MASK is the same, as shown in Equation (8), then no extra information about K_S can be obtained, as long as a strong MAC is used. It is assumed that the adversary does not know the long term key K_{AS} .

$$[A, B, K_S]_{K_{AS}} = [A, B, K'_S]_{K_{AS}} \quad (8)$$

Since B does not initiate the protocol, it has no input into the creation of MASK. So an adversary who pretended to be B has less input in the value of MASK than if they pretended to be A. The integrity of the key is also assured since key modification requires simultaneous modification of AUTH as well as $MASK \oplus K_S$.

This of course will fail if either A or B become compromised. Key establishment protocols between A and B cannot detect if either one of the sensor nodes is compromised. Communication between the nodes will need to be analyzed to detect if any false data has been sent [25], [27]. If the server becomes compromised, the key K_S may also become compromised. A possible solution to this is to use multiple servers to create the key, so that even if one or more servers become compromised it will not affect the security of K_S .

The key K_S can be used in the future to create or renew a session key between A and B. However, that relies on the assumption that the key K_S has not been compromised. Since K_S is never used as a session key or used to encrypt any plaintext, the keys K_{AS} and K_{BS} should be compromised before K_S . The sensor nodes should regularly refresh K_{AS} , K_{BS} and K_S with server S.

However, we should not discount that there may be some environments where K_S is more likely to be compromised before K_{AS} and K_{BS} . In this case, K_S should be refreshed more often than K_{AS} and K_{BS} . The natural thought will be to run our protocol from the start. However, there is an attack where an adversary can replay a portion of a previous message, so that the new key is still K_S . Note that there is no way to guarantee that the encryption of A, B, K_S is an old value or a new value. This type of attack can be classified as a replay attack. If the keys between the nodes and the KDCs are updated more frequently than K_S , then our protocols can be run

from the start without fear of a replay attack. The different keys between the nodes and the KDCs stop any adversary performing a replay attack.

Boyd's defense against the replay attack relies on a revocation list being available to all parties, but having a revocation list infrastructure may not be feasible in a low bandwidth, low energy, and low computational environment. We have therefore developed a fix to the protocol involving the addition of a short extra message at the start of the protocol, and having the nonces as part of the encrypted messages, as shown in *Proposed Protocol 2*. The construction of the *AUTH* now includes the nonces. Both nonces are added for the strongest security [39].

$$\begin{aligned} AUTH_A &= [A, B, N_A, N_B, K_S]_{K_{AS}} \\ AUTH_B &= [A, B, N_A, N_B, K_S]_{K_{BS}} \end{aligned}$$

Proposed Protocol 2 Fix to Boyd key agreement protocol

<i>M1</i>	$A \rightarrow B :$	A, N_A
<i>M2</i>	$B \rightarrow S :$	A, B, N_A, N_B
<i>M3</i>	$S \rightarrow B :$	$AUTH_B, MASK_B \oplus K_S$
<i>M3'</i>	$S \rightarrow A :$	$AUTH_A, MASK_A \oplus K_S, N_B$
<i>M4</i>	$B \rightarrow A :$	$[N_A]_{K_{AB}}$
<i>M5</i>	$A \rightarrow B :$	$[N_B]_{K_{AB}}$

D. Comparison with Existing KDC Mechanisms for Sensors

We will compare our protocols with existing KDC WSN protocols, such as SPINS [17] and PIKE [24]. Other KDC WSN protocols, such as TUTWSN KDC protocol [40], do not meet the criteria of having less than five messages and are therefore not considered. The SPINS protocol is described in *Protocol 5*. It contains four messages, with *S* (the KDC) sending two of those messages. The KDC generates K_{AB} and sends the key to both *A* and *B*. The nonces N_A and N_B are used to stop any replay attacks. Neither *A* nor *B* confirms that the other node has the correct key. Whereas, *Proposed Protocol 1* has key confirmation for both *A* and *B*.

Protocol 5 SPINS key agreement protocol

<i>M1</i>	$A \rightarrow B :$	A, N_A
<i>M2</i>	$B \rightarrow S :$	$N_A, N_B, A, B, [N_A, N_B, A, B]_{K_{BS}}$
<i>M3</i>	$S \rightarrow A :$	$[[K_{AB}]]_{K_{AS}},$ $[N_A, B, [[K_{AB}]]_{K_{AS}}]_{K_{AS}}$
<i>M3</i>	$S \rightarrow B :$	$[[K_{AB}]]_{K_{BS}},$ $[N_A, B, [[K_{AB}]]_{K_{BS}}]_{K_{BS}}$

Protocol 6 is the description of the PIKE protocol. This protocol contains three messages, with each node sending only one message. Sensor node *A* generates K_{AB} , which sends the key to *S*, which in turn sends it to *B*. Sensor node *A* can confirm that *B* does have K_{AB} , however, sensor node *B* cannot confirm that *A* has K_{AB} .

The key agreement protocol described in PIKE suffers from a replay attack. When *A* needs to refresh K_{AB} , then an adversary *C* can intercept or even initiate the protocol

Protocol 6 PIKE key agreement protocol

<i>M1</i>	$A \rightarrow S :$	$\{A, B, K_{AB}\}_{K_{AS}}$
<i>M2</i>	$S \rightarrow B :$	$\{A, B, K_{AB}\}_{K_{BS}}$
<i>M3</i>	$B \rightarrow A :$	$\{A, B, N_B\}_{K_{AB}}$

with *S* and *B*, as shown in *Attack 1*. This protocol is vulnerable to the replay attack since the messages *M1* and *M2* do not contain any nonces or timestamps to indicate that they are new messages. It is further compounded by the fact that *B* is not being able to confirm that *A* has the same key as *B* does. Another problem with the PIKE protocol is the lack of session identifiers. This has the effect of only having one running instance of the protocol at a time. The other protocols, described in this paper, have nonces, which can be used as session identifiers. Hence, they do not suffer from this issue. Because of the major limitations of the KDC protocol described in PIKE, we will not investigate it any further here.

Attack 1 Attack on the PIKE key agreement protocol

<i>M1</i>	$C \rightarrow S :$	$\{A, B, K_{AB}\}_{K_{AS}}$
<i>M2</i>	$S \rightarrow B :$	$\{A, B, K_{AB}\}_{K_{BS}}$
<i>M3</i>	$B \rightarrow C :$	$\{A, B, N_B\}_{K_{AB}}$

Since the SPINS protocol does not have key confirmation, when we compare the protocols, we will remove the key confirmation messages from the other protocols. In our *Proposed Protocol 1* we remove *M3* and *M4*, and messages *M4* and *M5* are removed from *Proposed Protocol 2*. In Bellare–Rogaway MAP1 protocol (*Protocol 4*) we can remove *M3*. The message *M2* will need to be modified to remove the key confirmation segment $[B, A, N_A, N_B]$ in the message.

Table III is a comparison of the SPINS protocol, *Proposed Protocol 1*, *Proposed Protocol 2* when keys are updated. The *m* value is the number of iterations of the protocol. Since our proposed protocols are built on top of the Boyd protocol and the *Modified Protocol 1*, and there are significant advantages in using the proposed protocols, we have not shown the Boyd protocol and the *Modified Protocol 1* in the table. Also, our comparisons, shown in the table, does not contain the performance impact of key confirmation between *A* and *B*. We also assume that the *MASK* attribute in the *Proposed Protocol 1* is calculated via an encryption algorithm, rather than a MAC. If our proposed protocols were using a MAC then it would be necessary to move the encryption values down to the MAC row. The nonces are simple counters, and the random numbers shown in this table is for the creation of any new keys. A sensor node sending data is a very expensive operation: each bit transmitted consumes the same amount of energy as 5000 instructions if the distance between the nodes is ten meters [41]. A useful aspect in our proposed protocols is that the key K_{AB} can be refreshed using the MAP1 protocol.

We next compare the number of cryptographic operations when performing multiple updates of key K_{AB} . For

TABLE III.
PERFORMANCE COMPARISON OF SINGLE SERVER PROTOCOLS

Protocol	SPINS	Proposed 1	Proposed 2
Messages sent	$4m$	$2m + 1$	$2m + 2$
En(de)cryption	$4m$	4	4
MACs	$6m$	$2m + 4$	$2m + 4$
PRFs	m	1	1
Bytes sent	$37m$	$4m + 28$	$4m + 32$

simplicity we assume that each operation costs the same amount. In the SPINS protocol there is $11m$ operations, where m is the number of iterations of the protocol. The number of cryptographic operations performed by our proposed protocol is $2m + 9$. If we also included key confirmation then our proposed protocols will have $6m + 9$ operations. Our proposed protocols has slightly more operations than the SPINS protocol. However, after only one key refresh, our proposed protocols perform fewer operations than if using the SPINS protocol.

The next comparison is the number of messages sent by each protocol. The SPINS protocol has a total of $4m$ messages, where m is the number of iterations of the protocol. The number of messages for *Proposed Protocol 1* with key confirmation is $3m + 2$ and without key confirmation the number of messages is $2m + 1$. Even with the higher number of messages found in key confirmation the *Proposed Protocol 1* has fewer messages sent after the second iteration. The number of messages for *Proposed Protocol 2* with key confirmation is $3m + 3$ and the number of messages without key confirmation is $2m + 2$. Once again, even with key confirmation the proposed protocol has fewer messages sent after only the second iteration.

The number of messages sent is not a true indication of the performance of the protocol, since the number of bytes sent is not considered. When investigating the amount of bytes sent, we found that the SPINS protocols has a total of $37m$ bytes, where m is the number of iterations of the protocol. The number of bytes for *Proposed Protocol 1* with key confirmation is $12m + 29$ and number of bytes without key confirmation is $4m + 28$. The number of bytes sent by the proposed protocol with key confirmation is significantly less on the second iteration. If there was no key confirmation, then the proposed protocol will have less bytes sent after the first run. The same test was performed for *Proposed Protocol 2*, and similar results were obtained. The number of bytes for *Proposed Protocol 2* with key confirmation is $12m + 32$ and the number of bytes without key confirmation is $4m + 32$.

E. Simulation Results

We used the values provided by the TOSSIM simulator (a part of the TinyOS installation) to obtain an indication of the power consumption when sending a message. In our calculations we do not take into account any collision avoidance times. On the mica2 mote, the cost of sending a 0 byte message is 9.9 microjoules, the cost of sending an 8

byte message is 21.1 microjoules, and a 16 bytes message is 32.4 microjoules. There is a substantial startup cost for each message sent, and then there is an added cost for every bit that is sent.

A memory comparison for our application was performed in a TinyOS environment. We compared CBC-MAC using SkipJack and RC5, and HMAC-MD5 on our application, as shown in Table IV. The ROM memory is greater for HMAC-MD5 than it is for CBC-MAC. The amount of RAM used is less for HMAC-MD5.

TABLE IV.
MEMORY OVERHEAD IN BYTES ON MICA2 PLATFORM

Memory	CBC-MAC (SKIPJACK)	CBC-MAC (RC5)	HMAC MD5	No Encrypt
ROM	13884	13290	24534	11532
RAM	617	617	592	528
.data	80	80	144	80
.bss	537	537	448	488
.text	13804	13210	24390	11452

The combination of *.bss* and *.data* segments use SRAM, and the combination of *.text* and *.data* segments use ROM. The values in Table IV indicate that HMAC-MD5 uses less RAM and more ROM than CBC-MAC. The *.text* contains the machine instructions for the application. The *.bss* contains uninitialized global or static variables, and the *.data* section contains the initialized static variables.

On further investigation the HMAC code itself only adds 176 bytes of *.text* and has no impact on the *.data* segment. The major memory cost in HMAC-MD5 is the cost of MD5, it uses a total of 12826 bytes. Further research into reducing the size of MD5, or investigating other one-way functions could lead to more memory efficient implementations. For instance, the *.text* section of HMAC-MD5 can be decreased if the common transformation macros are re-factored into functions.

We found that the TOSSIM simulator cannot be used for obtaining very accurate results when comparing the different MAC algorithms. Instead, ATEMU was used to show the number of cycles performed by the different MAC implementations, as shown in Figure 2. From our initial implementations, the size of the digest does not affect the number of cycles by a significant amount. We notice that HMAC-MD5 is significantly larger than both SKIPJACK and RC5, and RC5 performs better than SKIPJACK. When we increased the digest size from four bytes to 8 bytes, there was no significant difference in the number of cycles. It should be noted that this is not an extensive survey of MAC functions. For instance, there is no requirement for the underlying hash function used by HMAC to be collision free (it needs to be a one-way function), which greatly increases the number of MAC functions that can be used. However, even with a naive approach to implementing MD5 we have shown that it is possible to use HMAC-MD5 when no encryption algorithms are available.

After a detailed analysis of the proposed protocols and the comparison with the existing protocols, we conclude

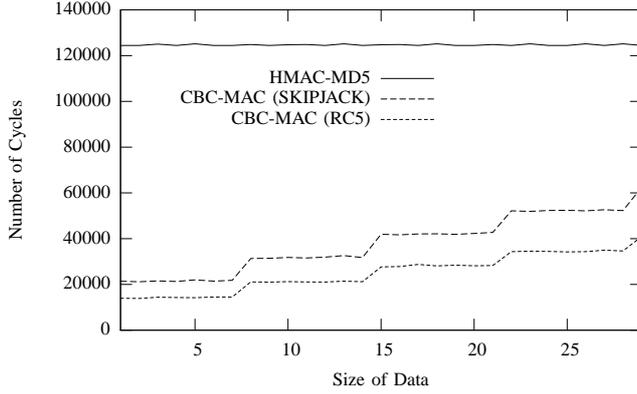


Figure 2. Number of cycles required for different MAC algorithms

that the large number of features (such as key confirmation, no need for large encryption algorithms, scalability, key freshness) found in *Proposed Protocol 1* and *Proposed Protocol 2* make them ideal candidates when creating a new multiple server protocol.

VI. NOVEL MULTIPLE SERVER PROTOCOLS

The single server protocols assume that server S is never compromised. In a sensor environment, this may not be always true. A solution is to use multiple authentication server protocols, specifically designed for the sensor environment. The multiple authentication server protocols we have developed are based on the concept of the Boyd four-pass protocol. These will maintain similar security characteristics as that of the centralized authentication protocol, as shown in the Boyd four-pass protocol. Because of the severe resource constraints which exist in sensor nodes, a multiple authentication server protocol should have low computational complexity in both time and space. A multiple server authentication protocol in a sensor environment should have the following characteristics:

- Small computation overhead,
- Minimal number of messages,
- Sensor nodes should not be relied upon to generate good randomness,
- The new key should be fresh,
- There should be key confirmation by both nodes.

A. Proposed Multiple Server Protocol with no encryption

In our attempt to create an efficient multiple server protocol, we specified n servers. The proposed protocol has the following message flows. The sensor node A sends the first message, A, B, N_A , to each of the servers. Each server sends their message to both sensor nodes A and B . Sensor node B sends N_B , the keying data, and the cross-checksums created by B . It is important to note at this stage that K_S is unknown, so unlike the original protocol, B is not able to send $[N_A]_{K_{AB}}$. When sensor node A receives the next message, it will calculate its own cross checksums, and compare them against the cross-checksums created by B . At this stage, the keys K_S

and K_{AB} are created. Sensor node A sends its cross-checksums to B , so B can create K_{AB} . The final message completes the key confirmation between A and B , as shown in *Proposed Protocol 3* [9].

The *Proposed Protocol 3* provides key authentication, key freshness and key confirmation, using multiple authentication servers. In our *Proposed Protocol 3*, the following constructs are used:

$$\begin{aligned} AUTH_{Ai} &= [A, B, K_i]_{K_{AS_i}} \\ MASK_{Ai} &= [[AUTH_{Ai}]_{K_{AS_i}}] \\ AUTH_{Bi} &= [A, B, K_i]_{K_{BS_i}} \\ MASK_{Bi} &= [[AUTH_{Bi}]_{K_{BS_i}}] \end{aligned}$$

Proposed Protocol 3 A Preliminary Multiple Server Protocol

$M1$	$A \rightarrow S_i$	A, B, N_A
$M2$	$S_i \rightarrow B$	$N_A, A, S_i, AUTH_{Bi},$ $MASK_{Bi} \oplus K_i$
$M2'$	$S_i \rightarrow A$	$S_i, AUTH_{Ai}, MASK_{Ai} \oplus K_i$
$M3$	$B \rightarrow A$	$cc_B(1), \dots, cc_B(n), N_B$
$M4$	$A \rightarrow B$	$cc_A(1), \dots, cc_A(n), [N_B]_{K_{AB}}$
$M5$	$B \rightarrow A$	$[N_A]_{K_{AB}}$

Both of the sensor nodes and the servers contribute to the key value. The values N_A and N_B are generated by A and B respectively as input to the MAC function, that determines the session key. The key used with the MAC function is generated by the servers. Both A and B compute the session key as $K_{AB} = [N_A, N_B]_{K_S}$. The nodes should have a minimum number of servers returning valid results before confirming that the key is valid. Node B will calculate $cc_B(i) \forall i \in 1, \dots, n$.

$$cc_B(i) = \begin{cases} [K_i]_{K_i} & \text{if valid,} \\ EM & \text{otherwise} \end{cases} \quad (9)$$

Where EM is an error message; an example will be the value zero. There is a remote chance a valid case may be zero. If the valid value is zero, the server needs to be considered a compromised server (even though it is not a malicious server).

Node A will calculate $cc_A(i)$, and compare it with $cc_B(i)$. If they are the same, then the server S_i is valid. Below is a way the nodes compare the cross checksum for $cc_A(i)$ and $cc_B(i)$.

$$cc_A(i) = \begin{cases} cc_B(i) = [K_i]_{K_i} & \text{if valid,} \\ EM & \text{otherwise} \end{cases} \quad (10)$$

After the comparison of the entire cross checksums, a set of valid keys V_1, \dots, V_m should remain. The creation of K_S is defined as follows.

$$K_S = V_1 \oplus \dots \oplus V_m \quad (11)$$

Where V_i is the i^{th} valid key given by a server, and m is the total number of valid servers $t \leq m \leq n$, where t is the minimal number of trusted servers. However, unlike the existing multiple server protocols, the trusted servers will not be able to calculate K_S . The calculated

$cc_A(i)$ values are returned to B , where B performs similar checks as A and calculates K_S .

B. Analysis of Proposed Protocol 3

Our *Proposed Protocol 3* has a number of advantages, one of which is that the nodes do not need good random number generators to create the nonces. The nodes could even safely use a counter for their nonce values. Another advantage is that if a server or a number of servers are unavailable, the authentication service itself still exists through the other servers. The servers and the sensor nodes have different keys; even if one or more servers become compromised, the authentication service or the security of the system is not compromised.

The proposed protocol only encrypts random information. If the encryption cipher uses an IV value (such as RC5 and SKIPJACK currently used in TinyOS [42]) then we can use a constant IV value. However, the constant IV value chosen for our protocol must only be used to encrypt the random data and should never be used to encrypt other information. Also, a wide variation of different ciphers can safely be used.

Some MACs have vulnerabilities when the message sizes are variable. All of our message sizes are of constant value, allowing us to safely use a wider range of MACs than previously available. The size of the MACs can be lower than that of conventional protocols. The integrity checking is performed by the sensor nodes. If x is the size of the MAC in bits, then an adversary has 1 in 2^x chance in blindly forging a valid MAC for a particular message. The adversary should be able to succeed in 2^{x-1} tries. Because of the low bandwidth of sensor nodes, a 4 byte MAC, requiring 2^{31} packets, will take years to complete. If an adversary did attempt this attack, the sensor node would be non-functional within that period. In addition, an adversary will need to forge $2t$ MACs; t MACs to A and t MACs to B , and stop traffic from the other base stations before they can determine the value of K_{AB} .

In order to study the performance impacts of each of the multiple server protocols, we first define the following symbols. The size of location indicator is a_0 , the nonce size is a_1 , the key size is a_2 , the hash size is a_3 , and the number of servers is n . The following equations are used to define how many bytes are sent for each message: $M1_i = 2a_0 + a_1$, $M2_i = 2a_0 + a_1 + a_2 + a_3$, $M2'_i = a_0 + a_2 + a_3$, $M3 = a_1 + na_3$, $M4 = (n + 1)a_3$, and $M5 = a_3$. However, there are n messages of type $M1$, $M2$ and $M2'$.

C. Reduction of Number of Messages

We have investigated and extended the *Proposed Protocol 3* further to reduce the large number of messages sent through the WSN. Thereby we developed the *Proposed Protocol 4*. We assume that the servers can communicate through a different network, other than the low bandwidth WSN used by the sensor nodes. For instance, the GNOME platform [12] also has an Ethernet connection it can

use as a high speed backbone network to communicate with other GNOME machines. In our *Proposed Protocol 4*, the sensor node A only sends one message to a server, denoted as S_1 . Server S_1 then gathers all the required information from the other servers through the server network (rather than the sensor network). Server S_1 concatenates the information and sends it to sensor node B . The list of servers may either be a static list, known by the sensor nodes, or it may be a list based on the trustworthiness of the servers.

Proposed Protocol 4 Reduced number of WSN messages

$M1$	$A \rightarrow S_1 :$	A, B, N_A
$M2$	$S_1 \rightarrow S_i :$	A, B
$M3$	$S_i \rightarrow S_1 :$	$MASK_{B_i}, AUTH_{B_i} \oplus K_i,$ $MASK_{A_i}, AUTH_{A_i} \oplus K_i$
$M4$	$S_1 \rightarrow B :$	$S_1, MASK_{B_1}, AUTH_{B_1} \oplus K_1,$ $\dots, S_n, MASK_{B_n},$ $AUTH_{B_n} \oplus K_n, N_A, A$
$M4'$	$S_1 \rightarrow A :$	$MASK_{A_1}, AUTH_{A_1} \oplus K_1, \dots,$ $MASK_{A_n}, AUTH_{A_n} \oplus K_n$
$M5$	$B \rightarrow A :$	$cc_B(1), \dots, cc_B(n), N_B$
$M6$	$A \rightarrow B :$	$cc_A(1), \dots, cc_A(n), [N_B]_{K_{AB}}$
$M7$	$B \rightarrow A :$	$[N_A]_{K_{AB}}$

1) *Security Analysis:* If S_1 becomes malicious, there are a number attacks that the server can try. The simplest attack is a denial of service, where the server will not gather any extra information from other servers, or doesn't respond to the sensor node B . If sensor node A does not receive a response from B in a required amount of time, then it should try S_2 .

Another possible attack S_1 can try is to forge the MACs to create legitimate messages from the other servers. As discussed earlier, it is unreasonable to assume a server can forge a message, let alone $2t$ messages.

If there are t malicious servers, then S_1 can contact those servers and not involve the trusted servers. However, the *Proposed Protocol 3* is also vulnerable to t malicious servers, as described in our explanation of *Proposed Protocol 3*.

If contacting only one server is still a concern, we can use a higher level reputation based framework [43] on top of existing authentication protocols to make sure the nodes collaborate with only trustworthy base stations. Another solution is that A can send the first message to p servers, where $p < n$, and each server is allocated servers from which to obtain information (however, this will require more code and logic within the sensor applications).

2) *Cost/Complexity analysis:* The *Proposed Protocol 4* decreases the number of packets sent by A . The following equations are used to define how many bytes are sent for each message, $M1 = 2a_0 + a_1$, $M2_i = 2a_0$, $M3_i = 2a_2 + 2a_3$, $M4 = a_0 + a_1 + na_2 + na_3$, $M4' = na_2 + na_3$, $M5 = a_1 + na_3$, $M6 = (n + 1)a_3$, and $M7 = a_3$.

Although the *Proposed Protocol 4* decreased the number of messages sent by A , it is not as reliable as *Proposed Protocol 3*. If S_1 is down, either A will need to detect this

and try S_2 , or S_1 itself will need to be a clustered system. The drawback of a true replicated clustered system is that it is more likely that the system can be compromised, since there are more machines available for an adversary to attack. Another problem if S_1 becomes malicious is that it may not return any information from the other servers. Once again, A can try other servers.

VII. SECURITY AND PERFORMANCE ANALYSIS

A. Security Analysis

The key K_S can be used in the future to create or renew a session key between A and B . However, K_S in the multiple server scenario has similar problems as the K_S in the single server scenario. The multiple server scenarios have strengthened the security between the nodes and the KDC. Compromised keys between a node and one (or more) servers, does not affect the security of the protocol. An adversary can replay previous (portion of) messages to force the sensor nodes to use the same K_S as before. If this is considered a concern, then we can convert *Proposed Protocol 2* to a multiple server protocol.

The *Proposed Protocol 5* is produced with the same methodology as the one used to produce *Proposed Protocol 4*. The advantage of the *Proposed Protocol 5* over *Proposed Protocol 4* is that if K_S is ever compromised, then the protocol can be run again safely. We modified the calculation of *AUTH* so that a replay attack is not possible.

$$\begin{aligned} AUTH_{A_i} &= [A, B, N_A, N_B]_{K_{A_i}} \\ AUTH_{B_i} &= [A, B, N_A, N_B]_{K_{B_i}} \end{aligned}$$

Proposed Protocol 5 Multiple Server Protocol with Secure Server Key

$M1$	$A \rightarrow B :$	A, N_A
$M2$	$B \rightarrow S_1 :$	A, B, N_A, N_B
$M3$	$S_1 \rightarrow S_i :$	A, B, N_A, N_B
$M4$	$S_i \rightarrow S_1 :$	$AUTH_{B_i}, MASK_{B_i} \oplus K_i,$ $AUTH_{A_i}, MASK_{A_i} \oplus K_i$
$M5$	$S_1 \rightarrow B :$	$AUTH_{B_1}, MASK_{B_1} \oplus K_1, \dots,$ $AUTH_{B_n}, MASK_{B_n} \oplus K_n$
$M5'$	$S_1 \rightarrow A :$	$AUTH_{A_1}, MASK_{A_1} \oplus K_1, \dots,$ $AUTH_{A_n}, MASK_{A_n} \oplus K_n$
$M6$	$B \rightarrow A :$	$cc_B(1), \dots, cc_B(n), N_B$
$M7$	$A \rightarrow B :$	$cc_A(1), \dots, cc_A(n), [N_B]_{K_{AB}}$
$M8$	$B \rightarrow A :$	$[N_A]_{K_{AB}}$

Proposed Protocol 5 has one more message than *Proposed Protocol 4*, however, *Proposed Protocol 4* has a larger message. This is because of the need to send A and N_A in $M4$, whereas in *Proposed Protocol 5* the message $M5$ does not need to send this extra information. The maximum size message in *Proposed Protocol 4* can be decreased if another message $M1'$ is sent as shown in Equation (12).

$$A \rightarrow B : A, N_A \quad (12)$$

B. Performance Analysis

When looking at authentication algorithms, a number of different aspects need to be taken into consideration. Apart from the security properties, such as key establishment, key freshness, and key confirmation, a number of performance aspects should also be looked at. In the past, symmetric key algorithm performance was categorized by the number of messages and the number of rounds. However, in sensor networks these are not true indicators of the performance of an algorithm. Other measures such as computational costs, number of bytes and packets sent and received, the amount of memory consumed are also important in a sensor environment.

The computational costs to our scheme arises because of the encryption, decryption and integrity checking of the keys generated by the servers. There is also the generation of the K_{AB} by both A and B . An inefficient aspect of the Boyd four-pass protocol is the need to decrypt the message before the integrity check is done. Our protocols have a similar restriction if bits were changed in the *MASK* then it will not be known until both *MASK* and *AUTH* were calculated. Bogus messages injected by a third party cannot be detected using any computationally efficient methods. The Bellare–Rogaway uses the encrypted messages when performing the integrity check, rather than the decrypted message. A very simple modification to the Boyd four-pass protocol removes this minor limitation, as shown in the *Modified Protocol 2*.

Modified Protocol 2 Boyd protocol integrity change

$M1$	$A \rightarrow S :$	A, B, N_A
$M2$	$S \rightarrow B :$	$[[K_S]]_{K_{AS}}, [A, B, [[K_S]]_{K_{AS}}],$ $[[K_S]]_{K_{BS}}, [A, B, [[K_S]]_{K_{BS}}]_{K_{BS}},$ N_A, A
$M3$	$B \rightarrow A :$	$[[K_S]]_{K_{AS}}, [A, B, [[K_S]]_{K_{AS}}]_{K_{AS}},$ $[N_A]_{K_{AB}}, N_B$
$M4$	$A \rightarrow B :$	$[N_B]_{K_{AB}}$

The size of the messages does not change, and there is the added benefit of the integrity check performed before the decryption of the message. We can also extend this technique to be used in our protocols (however, encryption algorithms are then required to be implemented on the sensor nodes). However, it has been shown that the communication costs are almost an order of magnitude more than the computational costs in security systems [17].

The communication costs will be heavily dependent on the topology of the network. Also, different protocols have different communication overheads for the sensor nodes, and the servers. The communication overheads of the different protocol is examined in detail in Section VIII.

VIII. COMPARISON

We now compare our proposed protocols with a number of the existing protocols. The computational complexity of the existing multiple server protocols is greater since

the key is constructed using a key-sharing mechanism. This has two major drawbacks in a sensor network environment. The first is the amount of extra code (and therefore memory overhead) needed when creating the new key. The second is the additional computation (and therefore extra energy) required when creating the new key. If we use a threshold scheme such as Shamir's scheme [30] to recover the key, the computational complexity will be $O(t \log_2 t)$. Whereas our proposed protocols use a simple exclusive-or function (as describe in Equation (11)) to recover the key, which has a computational complexity of only $O(n)$. Not only does this save on computational cost, but also has the added benefit of requiring less code than a full-blown key-sharing threshold scheme. The advantage of using a full-blown key-share threshold scheme, is that t servers can calculate the new session key between A and B if they are the only servers involved in the protocol. However, for performance reasons we have not placed the same restriction on our proposed multiple server protocols.

Another comparison is the communication cost of our proposed protocols compared with the existing protocols. For simplicity, we will assume that the output of the one-way function used to calculate the cross-checksums in the existing multiple server protocols is the same size as the integrity function used in our proposed protocols. Although, as described earlier, for security reasons the one-way function in the existing multiple server protocols will need to be larger.

We will compare the total number of bytes sent by sensor node A for each of the existing multiple server protocols, and our *Proposed Protocol 3* and *Proposed Protocol 4*. However, similar calculations as the ones shown here can be used to calculate the impact on the base stations and sensor B .

The total number of bytes sent by sensor A in the Gong multi-server protocol is $n^2 a_3 + 2na_0 + na_2 + 2a_0 + 2a_1 + a_3$, which has a complexity of $O(n^2)$. The number of bytes sent by sensor A in the Chen et al. multi-server protocol is $n^2 a_3 + na_3 + 3a_0 + 3a_1 + a_3$. The above two cases have a complexity of $O(n^2)$. However, the number of bytes sent by sensor A in *Proposed Protocol 3* and *Proposed Protocol 4* are $2na_0 + na_1 + (n + 1)a_3$ and $2a_0 + a_1 + (n + 1)a_3$, respectively. Both of these messages have a complexity of $O(n)$.

Figure 3 compares the total number of bits sent out in the entire network, relative to the number of servers used. The Gong protocol is the most expensive, followed by the Chen protocol. Also, the protocols consider for this graph send all their messages over the sensor network, rather than sending some of the messages over a faster backbone network.

The existing multiple server protocols have message sizes of size $O(n^2)$, whereas our proposed protocols are $O(n)$. It should be noted that the existing multiple server protocols do have added functionality, where the trusted servers are able to calculate the key K_{AB} . However, for performance reasons we have not placed the same

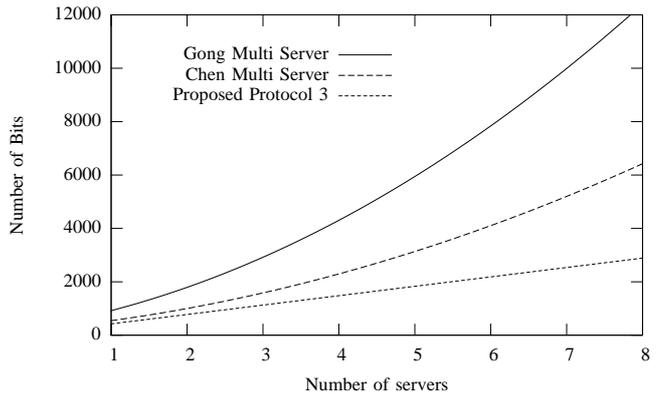


Figure 3. Total Number of bits sent by different Protocols

restriction on our proposed multiple server protocols.

Table V compares the three proposed multiple server protocols, where n is the number of servers. The table shows that in *Proposed Protocol 3* the performance hit is on sensor node A . The other two protocols put greater emphasis on the server nodes. The extra message in *Proposed Protocol 5* does not affect sensor node A , and for a large number of servers it is insignificant.

TABLE V.
PERFORMANCE COMPARISON OF PROPOSED MULTIPLE SERVER PROTOCOLS

<i>Proposed Protocol 3</i>				
Node	A	B	S_1	S_i
Msgs Tx	$n + 1$	2	2	2
Msgs Rx	$n + 2$	$n + 1$	1	1
Bits Tx	$72n + 32$	$32n + 40$	248	248
Bits Rx	$144n + 40$	$168n + 32$	40	40
<i>Proposed Protocol 4</i>				
Node	A	B	S_1	S_i
Msgs Tx	2	2	$n + 1$	1
Msgs Rx	3	2	n	1
Bits Tx	$32n + 72$	$32n + 40$	$232n - 40$	192
Bits Rx	$128n + 40$	$128n + 56$	$192n - 152$	40
<i>Proposed Protocol 5</i>				
Node	A	B	S_1	S_i
Msgs Tx	2	3	$n + 1$	1
Msgs Rx	3	3	n	1
Bits Tx	$32n + 72$	$32n + 88$	$240n - 48$	192
Bits Rx	$128n + 40$	$128n + 72$	$192n - 144$	48

The number of cryptographic operations performed by each node in the three proposed protocols shown in the table, is the same for each of the proposed protocols. The number of *AUTH* calculations is $2n + 4$ for both A and B . The number of *MASK* calculations is $2n$ for both A and B . For S_i the number of *AUTH* and *MASK* calculations is two. The only nodes that need to create good random numbers are each of the servers S_i and they only create one for each run through of the protocol. Both N_A and N_B can be a simple counter.

Another comparison is the number of packets sent by each of our multiple server protocols, as shown in Figure 4. The comparison is done over two different network topologies. The servers either communicate over

the sensor network or they have a separate network to communicate over. The *Proposed Protocol 3* has the same cost over both network topologies, since every message interacts with a sensor node, and the servers do not need to communicate with one another. Because of the larger number of messages sent by the *Proposed Protocol 4* and *Proposed Protocol 5*, it is natural for the number of packets to be more than *Proposed Protocol 3*, when all the messages have to be sent on the same sensor network. However, as the number of servers increases, the other proposed protocols converge with *Proposed Protocol 3*. The other proposed protocols have the advantage of concatenating messages, which the original proposed multiple server protocol did not have. If the servers can communicate over a different network, the number of packets sent by *Proposed Protocol 4* and *Proposed Protocol 5* is the same if the number of servers is equal to five. This is because of the first protocol having larger message sizes, and the messages need to be segmented sooner than they do in the other protocol. The number of packets sent on the energy-constrained sensor network is significantly larger if using *Proposed Protocol 3*, when the servers are on different networks.

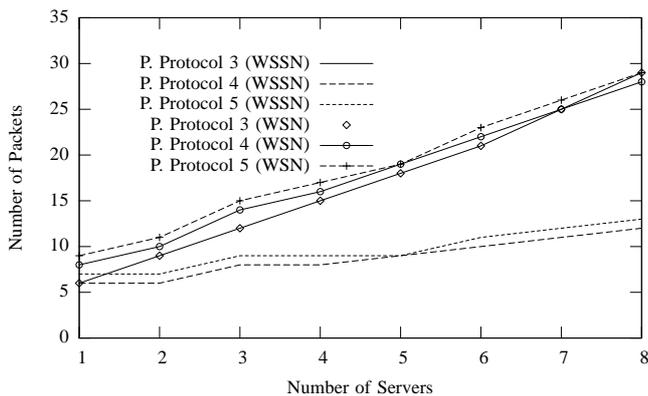


Figure 4. Number of packets sent by different Proposed Protocols

TinyOS has a seven byte overhead when sending a single packet. When comparing the bits sent by each of the protocols, we included the packet overhead. We once again cover the case of two network topologies, as shown in Figure 5. The *Proposed Protocol 3* has the same cost over both networks. When comparing the protocols, if the servers can communicate over a different network, we notice that if the number of servers is equal to five, there is virtually no difference between *Proposed Protocol 4* and *Proposed Protocol 5*.

Table VI compares the existing multiple server protocols with some of our proposed protocols. In the table we look at the server availability, and compare the protocols when the servers use the WSN or a different network. We also evaluate the robustness of the protocols based on security attacks in any environment.

After the detailed analysis of the proposed protocols and the comparison with existing protocols, we conclude that *Proposed Protocol 3* should be used in situations

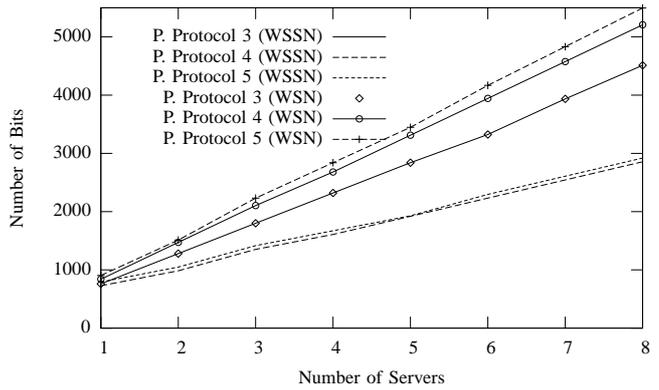


Figure 5. Number of bits sent by different Proposed Protocols

TABLE VI. COMPARISON OF MULTIPLE SERVER PROTOCOLS

Protocol	Properties of the Protocol			
	Server availability	Efficiency of WSN	Efficiency of WSSN	Security
Gong	Excellent	Bad	Bad	Excellent
Chen et al.	Excellent	Bad	Bad	Excellent
Proposed 3	Excellent	Good	Average	Good
Proposed 4	Average	Average	Excellent	Good
Proposed 5	Average	Average	Excellent	Excellent

where the environment is concerned with server availability. If the servers can communicate over a different network then *Proposed Protocol 4* should be used. If concerned about the security of K_S then *Proposed Protocol 5* should be used.

IX. CONCLUSIONS

Key distribution protocols, without the assumption of trusting an individual authentication server, are needed in sensor environments where clients cannot trust individual servers.

We examined the existing multiple server protocols developed for traditional networks, and investigated the problems using those protocols for WSNs. A critical review of single server protocols developed for both traditional and sensor networks was carried out. The performance of single server protocols, including two of our proposed protocols, was analysed in detail. We then proposed three multiple server protocols for sensor networks and surveillance sensor networks, and provided a detailed analysis and comparison of each the protocols.

The proposed multiple server protocols removed the requirement for the servers to know the keys between each of the sensor nodes, and thus helping to limit the message sizes to $O(n)$. The proposed protocols have the added benefit of not solely relying on the sensor nodes to generate cryptographically sound pseudo-random numbers, but still using information from each of the sensors to generate the new key. We have shown that our protocols are flexible enough for them to be used with almost any cryptographic primitive and in a range of environments.

REFERENCES

- [1] M. Kuorilehto, M. Hännikäinen, and T. D. Hämäläinen, "A survey of application distribution in wireless sensor networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 5, no. 5, pp. 774–788, 2005.
- [2] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *SensSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2004, pp. 162–175.
- [3] Axis, "Axis cameras and axis camera station software," <http://www.axis.com/>, 2007.
- [4] Genetec, "Omnicast – end-to-end ip video surveillance solution," <http://www.broadware.com/>, 2007.
- [5] P. Kulkarni, D. Ganesan, P. Shenoy, and Q. Lu, "Senseye: a multi-tier camera sensor network," in *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*. New York, NY, USA: ACM Press, 2005, pp. 229–238.
- [6] L. Gong, "Increasing availability and security of an authentication service," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, pp. 657–662, June 1993.
- [7] L. Chen, D. Gollmann, and C. J. Mitchell, "Key distribution without individual trusted authentication servers," in *8th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1995, pp. 30–36.
- [8] C. Boyd, "A class of flexible and efficient key management protocols," in *CSFW '96: Proceedings of the Ninth IEEE Computer Security Foundations Workshop*. Washington, DC, USA: IEEE Computer Society, 1996, p. 2.
- [9] K. Singh and V. Muthukkumarasamy, "A minimal protocol for authenticated key distribution in wireless sensor networks," in *ICISIP '06: Proceedings of the 4th International Conference on Intelligent Sensing and Information Processing*. Bangalore, India: IEEE Press, December 2006, pp. 78–83.
- [10] N. Bulusu, "Introduction to wireless sensor networks," in *Wireless Sensor Networks: A Systems Perspective*, N. Bulusu and S. Jha, Eds. Artech House, 2005.
- [11] Crossbow, "Crossbow," <http://www.xbow.com/>, 2006.
- [12] GNOME, "A controller node," <http://cmlab.rice.edu/projects/sensors>, 2006.
- [13] Medusa, "The medusa mk-2 controller node," <http://nesl.ee.ucla.edu/>, 2006.
- [14] MANTIS, "The mantis controller node," <http://mantis.cs.colorado.edu/>, 2006.
- [15] J. Deng, R. Han, and S. Mishra, "Sensor-network security, privacy, and fault tolerance," in *Wireless Sensor Networks: A Systems Perspective*, N. Bulusu and S. Jha, Eds. Artech House, 2005.
- [16] J. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless sensor network security : A survey," 2006, <http://www.cs.wayne.edu/~weisong/papers/walters05-wsn-security-survey.pdf>.
- [17] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," in *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM 2001)*, Rome, Italy, July 2001.
- [18] R. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus, "Tinypk: securing sensor networks with public key technology," in *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. New York, NY, USA: ACM Press, 2004, pp. 59–64.
- [19] D. J. Malan, M. Welsh, and M. D. Smith, "A public-key infrastructure for key distribution in tinys based on elliptic curve cryptography," in *Proc. 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communica-*
- [20] K. Singh, K. Bhatt, and V. Muthukkumarasamy, "Protecting small keys in authentication protocols for wireless sensor networks," in *Proceedings of the Australian Telecommunication Networks and Applications Conference*, Melbourne, Australia, December 2006, pp. 31–35.
- [21] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2003, p. 197.
- [22] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-secure key distribution for dynamic conferences," *Lecture Notes in Computer Science*, vol. 740, pp. 471–486, 1993. [Online]. Available: citeseer.ist.psu.edu/blundo95perfectlysecure.html
- [23] F. T. Leighton and S. Micali, "Secret-key agreement without public-key cryptography," in *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1994, pp. 456–479.
- [24] H. Chan and A. Perrig, "PIKE: Peer intermediaries for key establishment in sensor networks," in *Proceedings of IEEE Infocom*. IEEE Computer Society Press, Mar. 2005.
- [25] B. Przydatek, D. Song, and A. Perrig, "Sia: secure information aggregation in sensor networks," in *SensSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 255–265.
- [26] D. Wagner, "Resilient aggregation in sensor networks," in *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. New York, NY, USA: ACM Press, 2004, pp. 78–87.
- [27] S. Zhu, S. Setia, and S. Jajodia, "Leap: efficient security mechanisms for large-scale distributed sensor networks," in *SensSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 308–309.
- [28] J. Deng, R. Han, and S. Mishra, "Intrusion tolerance and anti-traffic analysis strategies in wireless sensor networks," in *Dependable Systems and Networks, 2004 International Conference on*. IEEE, June 2004, pp. 637–646.
- [29] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, U. Maurer and R. Rivest, Eds. Springer Berlin / Heidelberg, 2003.
- [30] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [31] R. K. Bauer, T. A. Berson, and R. J. Feiertag, "A key distribution protocol using event markers," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, pp. 249–255, 1983.
- [32] D. Otway and O. Rees, "Efficient and timely mutual authentication," *SIGOPS Oper. Syst. Rev.*, vol. 21, no. 1, pp. 8–10, 1987.
- [33] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *Proceedings of the Royal Society of London*, vol. A426, pp. 233–271, 1989.
- [34] M. Bellare and P. Rogaway, "Provably secure session key distribution: the three party case," in *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1995, pp. 57–66.
- [35] M. Abadi and R. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Trans. Softw. Eng.*, vol. 22, no. 1, pp. 6–15, 1996.
- [36] ISO11770-2, "Information technology — security techniques — key management — part 2: Mechanisms using symmetric techniques iso/iec 11770-2," 1996.
- [37] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *CRYPTO '93: Proceedings of the 13th*

- annual international cryptology conference on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1994, pp. 232–249.
- [38] P. Janson and G. Tsudik, “Secure and minimal protocols for authenticated key distribution,” *Computer Communications*, vol. 18, no. 9, pp. 645–653, September 1995.
- [39] K.-K. R. Choo, C. Boyd, and Y. Hitchcock, “On session key construction in provably-secure key establishment protocols,” in *Mycrypt*. Springer, 2005, pp. 116–131.
- [40] P. Hämäläinen, M. Kuorilehto, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, “Security in wireless sensor networks: Considerations and experiments,” in *SAMOS*, 2006, pp. 167–177.
- [41] R. Govindan, “Wireless sensor network tutorial,” <http://www.comsoc.org/freetutorials/nsc/>, 2006.
- [42] TinyOS, “An operating system for sensor motes,” <http://www.tinyos.net/>, 2007.
- [43] S. Ganeriwal and M. B. Srivastava, “Reputation-based framework for high integrity sensor networks,” in *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. New York, NY, USA: ACM Press, 2004, pp. 66–77.

Kalvinder Singh is a developer for Australia Development Labs, IBM, and is also a PhD student in the School of Information and Communication Technology at Griffith University, Gold Coast, Queensland, Australia. He received a BSc(Hons) with a university medal from James Cook University, Townsville, Queensland, Australia. His research interests include identifying efficient security protocols for wireless sensor networks.

Vallipuram Muthukkumarasamy received the BScEng (Hons) from the University of Peradeniya, Sri Lanka in 1984 and the PhD from the Cambridge University, England in 1990. He is currently a Senior Lecturer in the School of Information and Communication Technology at Griffith University, Gold Coast, Australia. His current research interests include intrusion detection and prevention techniques, secure key establishment in wireless sensor networks, preventing denial of service attacks in 802.11 networks, and security and privacy issues in e-government.