2012 IEEE 11th International Symposium on Network Computing and Applications

DiFiSec: An Adaptable Multi-level Security Framework for Event-driven Communication in Wireless Sensor Networks

Syed Rehan Afzal*, Christophe Huygens†, Wouter Joosen†
*Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
†IBBT DistriNet Research Group, Department of Computer Science, Katholieke Universiteit Leuven, Belgium
Email: s.r.afzal@tue.nl, christophe.huygens@cs.kuleuven.be, wouter.joosen@cs.kuleuven.be

***Abstract*—**State of the art security research in the field of wireless sensor networks has focused on providing security in a coarse-grained, full-fledged and static fashion. This implies providing confidentiality, data authentication, data integrity and freshness to the entire spectrum of communication between the participating nodes in a network. In this paper however we advocate that as a result of a number of factors relating wireless sensor networks, providing security in similar fashion for the entire communication set isn't a pragmatic approach and does not precisely reflect the application level security requirements. We therefore propose *DiFiSec*, a dynamic, fine-grained and adaptable security framework that supports various levels of pluggable security for distinct data communication sets depending on the context, environment and criticality of the data. These pluggable security levels can be enacted at the levels of component wirings and receptacles, hence empowering application users to select only the most appropriate security respecting the resource-constrained nature of WSNs. Furthermore, to support system evolution and changing application requirements DiFiSec offers runtime adaptability. A prototype of this system has been implemented on SunSPOT sensor nodes where we have evaluated our approach in comparison with other network security variations.

I. Introduction

Perrig et. al. [2] suggested in their work that wireless sensor networks (WSNs) require four distinct security attributes i.e. data confidentiality, data authentication, data integrity and data freshness in order to make communication between the participating *nodes* completely secure against potential threats. In our paper, we call such an approach a *coarse-grained, full-fledged* and *static* security approach. Communication in WSNs can primarily be classified into control packets and data packets communication. In the context of wireless communication, control packet communication comprises management packet that may relate to routing maintenance, error control, policy updates and mobile code etc. On the other hand data packets may involve event data sensed and disseminated from a publishing node to the subscribing devices. In an environment marked by scarcity of resources in terms of processing power, energy, signal strength etc. it isn't feasible to implement a full-fledge security for the entire communication stream. As a consequence, real life implementations opt for either no security or merely securing the Control Communication. On the other hand, an *all or nothing* security approach concerning the Data Communication results in a strenuous tradeoff regarding the application stakeholders. Just as traditional courier services mark items as ordinary, normal or fragile and ensures appropriate handling respectively; we present a security framework that equips stakeholders to *security tag* Data Communication on merit e.g. criticality of data.

In this paper, we make two contributions in the area of security in WSNs. First, we outline the factors aided with application scenario that advise against a static, full-fledged and coarse-grained security framework. Secondly we present the design and evaluation of our proposed security framework *DiFiSec*, which to the best of our knowledge is the first work to achieve selective multi-level security provisioning on data between fine-grained communicating components that supports runtime adaptability. DiFiSec encompasses four categories of communication i.e. *DiFiSec Event Subscription Request (DiFiReq)*, *Subscription Reply (DiFiRep)*, *Event Dissemination* and finally *Application Security Level (ASL) Adaptation*. It enables applications to apply varied levels of pluggable security to distinct data flows depending on the context, environment and criticality of the information. These pluggable security levels can be enacted at the levels of component wirings and receptacles. Furthermore, to support system evolution and changing application security requirements DiFiSec supports adaptability to re-adjust the security level bindings at the runtime.

In what follows, Section 2 summarizes the related work while Section 3 outlines the design goals for our framework. Section 4 sets the pitch with a motivational scenario illustrating a harbor shipment management scenario. Furthermore, we detail our security framework *DiFiSec* in Section 5. Section 6 demonstrates the performance gain from our DiFiSec implementation on SunSPOT [9] sensor platform. Finally in Section 7 we present our conclusions.

II. Related Work

A number of secure sensor network protocols have been proposed including TESLA [4], SPINS [2], LISP [7], LEAP [1] and TinySec [14]. Besides inter node and broadcast confidentiality, secure key management & distribution etc.; these schemes also offer some level of access control, entity authentication and non-repudiation.

SPINS [2] is based on Sensor Network Encryption Protocol (SNEP) and μTESLA (a minimal implementation of the TESLA [4] protocol). SNEP provides data confident-

978-0-7695-4773-2/12 $26.00 © 2012 IEEE
DOI 10.1109/NCA.2012.49

iality, two-party authentication, integrity and freshness whereas µTESLA provides authentication for data broadcast. Both of these security schemes are used with a shared secret key to send packets between nodes. µTESLA emulates asymmetry through a delayed disclosure of symmetric keys and serves as the broadcast authentication service of SNEP. It is done employing loosely time synchronized timer on both the base station and other nodes to authenticate the MAC key. µTESLA relies solely on this delayed disclosure, unlike its predecessor (TESLA), which authenticates the initial packet using digital signature. µTESLA phases are sender setup, broadcasting authenticated packets, bootstrapping a new receiver, and authenticating broadcast packets.

LiSP [7] Lightweight Security Protocol for Wireless Sensor Networks is developed as a security protocol with a tradeoff between security and resource consumption via efficient rekeying. LISP uses rekeying protocol that has 1) efficient key broadcasting without retransmission/ACKs, 2) implicit authentication for new keys without incurring additional overhead, 3) seamless key refreshment without disrupting ongoing data transmission and 4) robustness to inter node clock skews among nodes. The architecture of LiSP is based primarily on the use of symmetric keys based upon master keys (MK) and temporal keys (TK). Network nodes are segregated into groups with each group having an automatically selected key server (KS). LiSP provides a great deal of protection from compromised nodes and key servers. The keying system with implicit authentication allows the sensor to quickly detect whether or not the key that was sent from the key server is authentic or not.

LEAP [1] Localized Encryption and Authentication Protocol requires each node to store 4 different types of key. Every node has an individual key which is shared only with the Base Station (BS). Each node will also have a copy of the group key, which is shared by all the nodes on the system. This key is usually used for the BS to send out broadcasts to the nodes. The third key that a node has is the cluster key: a key shared by the node and its neighbors hence securing local broadcast messages. The final type of key is the pairwise shared keys. Each node will have a pairwise shared key for each of its neighbors. By using these different keys the messages sent can be protected to different extents. LEAP implements µTESLA [2] to enable broadcast messages from the BS to be secured. In order to authenticate a message the sending node signs it with its cluster key and transmits it. When this message is received by another node it is verified using the relevant cluster key, the data is then authenticated with its own cluster key and then forwarded on to the next node.

TinySec [14] is a security protocol comprising two security modes: Authenticated Encryption (TinySec-AE) mode and only Authentication (TinySec-Auth). The design goal of this project was to create a link layer security protocol that contains access control, message integrity, and confidentiality. TinySec uses CBC-MAC to authenticcate that the message has not been altered, but rather than using an 8 or 16 byte MAC TinySec uses 4 bytes. Unlike SPINS, LiSP and LEAP etc., TinySec, leaves the selection of the keying mechanism up to the application developers.

III. DESIGN GOALS

From the state of the art in WSN communication security in Section 2; we can notice that the aforementioned protocols achieve secure communication by using different authentication schemes comprising digital signatures, hashing functions and symmetric or asymmetric keying. Moreover, they combine timestamps, sequence numbers, initialization vectors, random number and tokens etc. to insure non-repudiation and data freshness. Each of these approaches has its own distinct traits; however, what they have in common is the *static nature* (except TinySec) and *level of coarseness* of these security schemes.

These schemes provide a level of entity authentication (coarse-grained) which considers nodes as access points e.g. node A is allowed or not allowed access to node B. However, we envision an architecture that provides a further fine grained correspondence between nodes, where the node owners could set policies authorizing access to only a particular set of components or interfaces; rather than the entire node. From an applications perspective we establish that different communication streams could have different security requirements. And therefore in a resource constrained environment it becomes vital to provide full-fledged security only for the communication where the application requires it fittingly. Furthermore since these requirements are subject to change, we endeavor to achieve a dynamic and adaptable system that supports runtime tweaking of communication streams with security levels. As a result of our related research and understanding of security requirements in the WSN domain, we derive the following four design goals for our framework.

A. *Lightweight*

The first and foremost design goal is the need for a secure communication framework to be lightweight. Traditional full-fledged and full-ON security prove to be too exhaustive to run for the entire spectrum of communication in a constrained environment as WSNs

B. *Fine-grained*

In state of the art secure communication frameworks, packets are treated at a coarse grained level of participating nodes i.e. sending node and receiving node(s) [18,2,7,1,13, 14,6]. We advocate a further fine-grained paradigm where communication can be classified between partic-ipating components, interfaces, events, nodes as well as networks. In order to achieve multilevel granularity, we have adopted for a solution which is event driven and adheres to component based paradigm [10, 8, 5].

C. *Flexible*

Furthermore, we envision an ideal communication model to be flexible in a sense that it allows and supports varied levels of security for different sets of communication. For instance in a conference hall building, administrators are equipped to set communication from TIME_DATA disseminating sensor components to only ensure integrity of its data since this information is not confidential and thus needs not be encrypted. In another case, it may be desired to have data coming from TEMPRATURE_MONITOR component to always ensure its confidentiality as well as integrity. It is therefore flexibility which helps achieving a runtime light-weight framework by avoiding unnecessary full-fledged security operations for the entire spectrum of communication and thus avariciously following precisely what the business logic advocates.

D. *Adaptive*

Our fourth and final goal is achieving adaptability. By adaptability we mean that our system has to have the ability to runtime adjust and tweak the security levels for any given communication stream resulting from a changing application security requirements. Further, in a broader perspective the adaptability design goal also ensures a system that requires limited runtime pre-configuration. In Section 4 and 6 we will show how our design goals of *fine-granularity*, *flexibility* and *adaptability* help us achieve a communication framework that is runtime *lightweight*.

IV. MOTIVATIONAL SCENARIO

We derive our motivation from a WSN aided tracking and monitoring system that monitors packages and containers for the harbor company as well as other involved stakeholders. Our scenario involves packages from two companies namely Fruit Trade Company (FC) owning nodes {*FC Node1, FC_Pickup*} and Pharmaceutical Company (PC) owning nodes {*PC_ Node1, PC_Node2 and PC_ Pickup*}. The Harbor Company (HC) owns a monitoring sensor node in each container which in this case disseminates Temperature, Location and Humidity data. Figure 1 below illustrates the business logic at a given time where:

- HC dictates the event data received from all nodes to preserve integrity, confidentiality and authorization.
- PC reckons only the temperature data's integrity is critical for the packages being transported. Hence the component wiring between HC.TEMPRATURE and PC_Node2.TEMPERATURE_MONITOR ensures only the data's integrity by using 1-way hashes.
- Location data is deemed critical from integrity and authentication perspective where involved parties need to ensure the data wasn't tampered as well as it came from the authentic source. Furthermore, the confidentiality of the data isn't required. Hence the component wiring involving Location data preserve integrity and authentication using Hash based Message Authentication Code (HMAC).
- To avoid any overhead PC_Node1.HUMIDITY_MONITOR component requires data from HC.HUMIDITY component to bypass all security primitives.

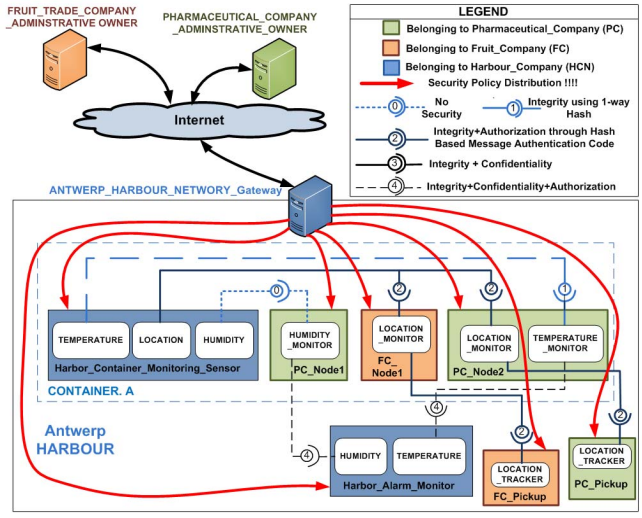

Figure 1. Harbor Shipment Management Scenario

V. DISTRINET FINE GRAINED SECURE COMMUNICATION FRAMEWORK (DIFISEC)

We propose our secure communication framework that is based on event driven communication paradigm. Our approach advocates the notion of component based, event driven binding model where the security features can be enacted and modified at run time. In general, communication in Wireless Sensor Networks constitutes control and data packets transmission. In a component based event driven paradigm, control packets translate to contracts/policy data between subscriber (client) and publisher (server), whereas data packets may include sensor readings in the form of event data transmitted from publisher to the subscribers. Our approach aims to offer multiple levels of pluggable security for different data communications depending upon the context, environment and criticality of the information.

A. *Pluggable Security Levels*

1) *Framework Apparatus*

Towards achieving pluggable security, we define the set of atomic security apparatus for DiFiSec. This comprises 1-way Hash functions, Message Authentication Code (secure-Hash-based or Symmetric-block-cipher-based depending on the keying mechanism), Symmetric/ Asymmetric Keys, Encryption/ Decryption, Digital Signatures and Initialization Vectors/ Time Stamps/ Pseudorandom Seeds.

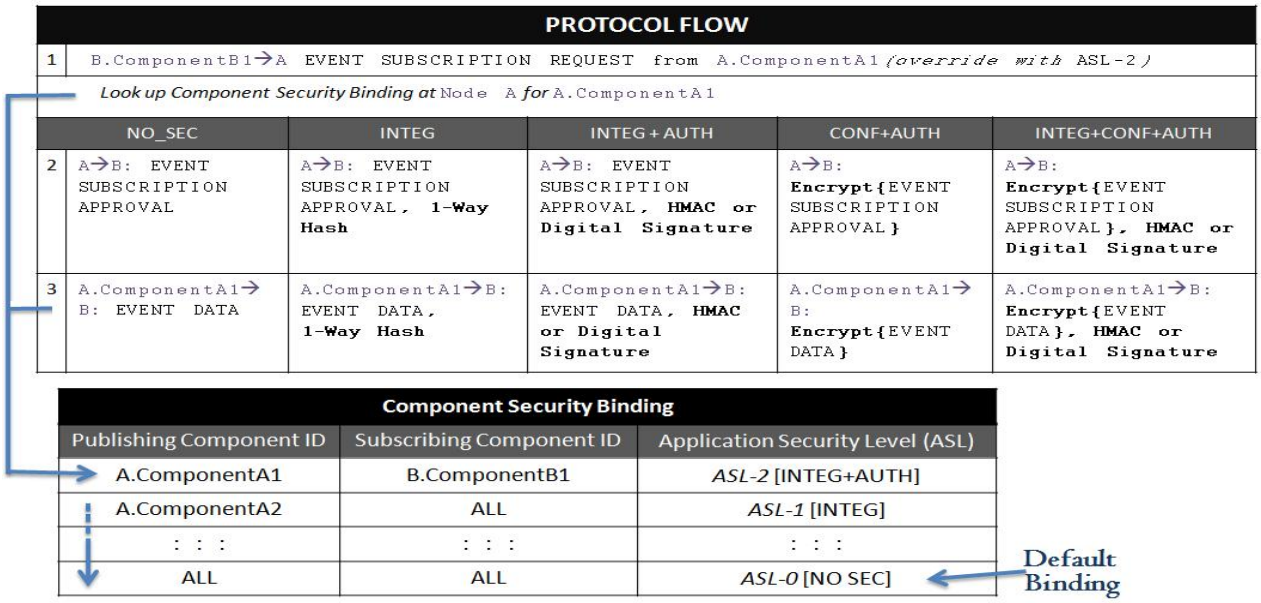

Figure 2. Basic Protocol Flow

2) Application Security Levels (ASL)

We suggest that data communication at any level in order to represent application security view point will involve any of the possibilities as outlined in Table 1.

ASL	NON-FUNCTIONAL SECURITY COMPOSITIONS
0	No SECURITY
1	INTEGRITY
2	INTEGRITY + AUTHENTICATION
3	CONFIDENTIALITY + AUTHENTICATION
4	INTEGRITY + CONFIDENTIALITY + AUTHENTICATION

Table 1 –Pluggable Security Level Definition

For DiFiSec Framework, out of the possible $|P(S)| = 2^n$ permutations of *ASL* levels, Authentication or confidentiality can't exist standalone. Authenticity in DiFiSec communication can be achieved by Digital Signatures thus providing integrity as well. Authentication can however be achieved standalone in sensor networks by mechanisms of access or resource control [17]. Since DiFiSec confidentiality achieves node authentication as well, an ASL level only enforcing 'Confidentiality + Integrity' isn't practical as well. Our framework is composed of the following:

- A loosely-coupled WSN component model that allows easy inspection of data flows between the components and component interfaces that compose applications.
- A flexible and extensible security engine which allows for fine grained control of data flows.
- A secure policy distribution framework that ensures only the authorized actors can deploy security policies on the nodes and when enacted, these policies could not be circumvented.
- Pluggable Security levels that can reflect the application security requirements at any given point.

B. Basic Protocol Flow

Figure 2 shows the general running of DiFiSec. A wiring between components is initiated by an *Event Subscription Request* from a Subscribing Node and may state the desired *ASL* for the wiring. If not, the publishing component chooses the default ASL. In the prior case, the node makes a new entry in the *Component Security Binding (CSB)* table. This table is present on all nodes and contains the customized and default ASL mappings of the components belonging to that particular node. In the case illustrated in the figure, Node A adds a new entry in the CSB Table that declares CSB between *A.ComponentA1* and *B.ComponentB1* as ASL-2. This enforces that the data communication between these two nodes shall be preserveng Data Integrity and Authentication. This Event Subscription Request is followed by the *Event Subscription Approval* packet from Node A. Hence at the time of Event Data dissemination, Node A looks up the CSB table and thus ensures ASL-2 security for this Event flow. This way DiFiSec couples component bindings to assorted security levels. As far as tweaking these security levels is concerned, adaptability can be initiated by one of the three stakeholders:

- Policy bytes coming from the backend adding, removing or modifying CSBs on a given node.
- Subscribing node's component requesting to change ASL of its wiring with publishing node's component.
- Publishing node's component requesting to change ASL of its wiring with subscribing node's component.

In either of these cases, the ASL change request goes to the Gateway which after authenticating the source redirects it to the destination. The mechanism employed to disseminate such policies securely from backend to participating nodes is described in our previous work [3].

C. Detailed Protocol Description

1) DiFiSec Event Subscription Request (DiFiReq)

DiFiSec wiring is initiated by an *Event Subscription Request* packet sent from the interested component's *Event Dispatcher* destined towards the publishing node. The packet format is as follows:

{DiFiReq_header, Source, Destination, Source_component ID, Destination_componentID, nonce, Ti , ASLi (optional), HM_0}
Where $HM_0 = MAC_K (M_0) = MACK$ (Source, Destination, Source_componentID, Destination_ componentID, nonce, Ti, ASLi)

Here Source and destination component IDs specify the components to be wired. *Nonce* and T_i are employed to avoid packet replays and ensure freshness of request. The *ASL* field is optional which if used, suggests the desired ASL to destination node. Finally H_{M0} is the computed Message Authentication Code of the request's payload using the secret key '*K*' shared between the source and destination. H_{M0} is thus computed by the *HMAC Engine* as shown in Figure 3. DiFiReq packet is then forwarded to the distribution channel through the *Event Dispatcher*.

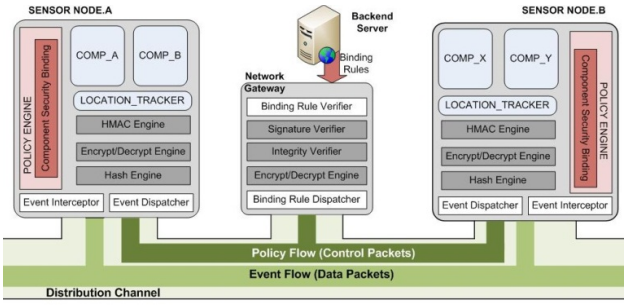

Figure 3. DiFiSec Deployment View

2) DiFiSec Subscription Reply (DiFiRep)

The DiFiReq packet once dispatched is directed to the destination via the Distribution Channel. Once the packet is received by the destination node it is caught at the destination node's *Event Interceptor*. The Event Interceptor looks at the header and recognizes it as a subscription request and thus extracts the MAC H_{M0} and passes it to *HMAC Engine* that verifies the packet integrity as well as authenticates it since the MAC is recomputed using the shared secret key between the two nodes. Once verification is successful, the Event Interceptor looks up for ASL field. At this point it can be one of the two cases:

- *The ASL information is explicitly sent by the publishing/subscribing node* and thus the Event Interceptor inserts a new entry in its node's CSB.
- *Participating nodes didn't send the ASL information* and thus the Event Interceptor does nothing. Hence, later on at the time of event dissemination the Event Dispatcher shall look up for a specific CSB entry or otherwise would follow the *Default Binding* (Fig. 2).

DiFiRep packet format is as follows:

{DiFiRep, Source, Destination Source_componentID, Destination_componentID, nonce, T_i , ASL^i (optional), H_{MI} }
Where $H_{MI} = MAC_K (M_I) = MAC_K$ (Source, Destination, Source_componentID, Destination_componentID, nonce, T_i)

The *ASL* field is optional which if used overrides any previously existing Component Security Binding between the participating components. Once the packet is constructed the Event Dispatcher forwards it to the subscription requesting node implying a subscription approval.

3) DiFiSec Event Dissemination

Publishing Nodes keep track of the list of subscribers for each of their components. Depending on the scenario event dissemination could be triggered for onetime, periodical or reactive to some other event(s). Once the event triggers, the corresponding component forwards the *Event_ Data* to the Event Dispatcher which looks up for the ASL for the participating component in the CSB table. Depending on the ASL^i where *i* is the level of security for the binding, the Event_Data payload is constructed following the appropriate packet format from Table 2.

ASLi	*PACKET FORMAT*
ASL-0	*{Event_header_ASL0, Source, Destination, Source_ componentID, Destination _componentID,Event_Data}*
ASL-1	*{Event_header_ASL1, Source, Destination, Source_ componentID, Destination _componentID,Event_Data, nonce, Ti , H_M}* *Where $H_M = h(M_0)$ = 1-way-hash (Source, Destination, Source _componentID, Destination_componentID, nonce, Ti, Event_Data)*
** ASL-2*	*{Event_header_ASL2, Source, Destination, Source _componentID, Destination _componentID, Event_Data, nonce, T_i , H_{MAC} } [a]* *Where $H_{MAC} = MAC_K (M_0) = MAC_K$ (Source, Destination, Source _componentID, Destination_componentID, nonce, T_i, Event_Data)*
ASL-3	*{Event_header_ASL3, Source, Destination, Source _componentID, Destination _componentID, nonce, T_i , E_M}* *Where $E_M = Encrypt_K (M_0) = Encrypt_K$ (Source, Destination, Source _componentID, Destination _componentID, nonce, T_i, Event_Data)*

ASL-4	*{Event_header_ASL4, Source, Destination, Source _componentID, Destination_componentID, nonce, T_i , E_M , H_{EM} }* *Where E_M = Encrypt$_K$ (M_0) = Encrypt$_K$ (Source, Destination, Source _componentID, Destination_ componentID, nonce, T_i, Event_Data)* *And H_{EM} = 1-way-hash (E_M) or MAC$_K$ (E_M)*

Table 2 –Packet Formation dictated by Application Security Level (ASL)

* For ASL-2, depending on the implementation HMAC could be replaced by Asymmetric Key based Digital Signature which would also achieve the message's integrity as well as sender's verification.

After the respective event packet is constructed, the Event Dispatcher forwards it to the Distribution Channel where it's eventually received at the subscribing node's Event Interceptor. The Event Interceptor inspects the header, recognizes the ASL of the packet and thus forwards it to the respective Engines.

4) DiFiSec ASL Adaptation

As stated earlier, ASL adaptability in the CSB table can be enacted by one of the three stakeholders i.e. publishing node, subscribing node and the backend. The publishing and subscribing nodes can advise recommended ASL at the binding time as well as during the normal runtime course. At binding time subscribing and publishing nodes can employ the *optional ASL* field in DiFiReq and DiFiRep respectively. While during the runtime ASL of a given wiring can be tweaked by an ASL adaptation request followed by an Acknowledgement from the other node. The packet format is as follows:

{Adaptation_request, Source, Destination, Source_ componentID, Destination _componentID, nonce, T_i , ASL^i, H_M}
Where H_M = MAC$_K$ (M) = MAC$_K$ (Source, Destination, Source _componentID, Destination_componentID, nonce, T_i, ASL^i)

The Adaptation Request is followed by an Acknowledgement with the following format:

{Adaptation_reply, Source, Destination, Source_ componentID, Destination _componentID, nonce, T_i , ASL^i, H_M}Where H_M = MAC$_K$ (M) = MAC$_K$ (Source, Destination, Source _componentID, Destination_componentID, nonce, T_i, ASL^i)

At this point the publishing and subscribing nodes update their CSB entry which is adhered to for the next event transmission by the publishing node's Event Dispatcher. As far as adaptation from the backend is concerned, we proposed the policy dissemination from Backend System via Network Gateways in our previous work in [30]. In the context of DiFiSec Security Framework the packet format would be:

{Adaptation_policy, Network_Gateway, Publishing_node ID, Subscribing_nodeID, Publishing_componentID, Subscribing_ componentID, nonce, T_i , ASL^i, $H_{M\Omega}$}
Where $H_{M\Omega}$ = MAC$_{K\Omega}$ (M) = MAC$_{K\Omega}$ (Source, Destination, Source _componentID, Destination_componentID, nonce, T_i, ASL^i)

Adaptation policy from the backend always has to come via the Network's Gateway. Therefore *Network _Gateway* refers to the address of the Gateway node. Then it states the participating components of the publishing and subscribing nodes as well as the new ASL. Finally $H_{M\Omega}$ refers to the Secure Hcomputed using the secret key K_Ω shared between the Backend Node and the Network Gateway. Once the Network Gateway receives the packet, it authenticates the Backend node as well as verifies the packets integrity and then reconstructs the packet as *Adaptation_request* packet shown earlier. Which when received by the publishing and subscribing nodes trigger corresponding CSB entry update.

VI. EVALUATION

We have evaluated the performance of DiFiSec Security Framework on Java ME CLDC 1.1 compliant SunSPOT nodes [9]. SunSPOTs are embedded hardware modules that are equipped with a 180 MHz ARM9 processor, 512 KB RAM, 4MB flash memory, three on-board sensors (temperature, light and three-axis accelerometer), hardware interfaces for the integration of arbitrary external sensors, and the IEEE 802.15.4 wireless transmission technology [15]. For our implementation we used the Squawk Virtual [21] Machine 'RED' version [12].

We employed DistriNet's Loosely-coupled Component Infrastructure (LooCI) [10] framework as the component based system. LooCI is a platform-independent specification of a component infrastructure middleware comprised of a reconfigurable component model, a hierarchical type system and a distributed event bus. The LooCI event bus implements a fully decentralized publish-subscribe interaction model, which is realized by the Event Manager component running on each node. At the time of writing LooCI supports Contiki [19], Squawk [21] and OSGi [20]. In our implementation we have implemented event management as a combination of Event Interceptor and Event Dispatcher. The Event Interceptor thus forms a universal point of interception for node's component interactions and an ideal point at which to implement *Component Security Bindings* for the incoming events. Furthermore, the Event Dispatcher ensures the Component Security Bindings for all the outgoing events. This equips application developers with no security background develop LooCI components completely oblivious to security and then merely state the wiring's non-functional security requirement as level 0,1,2, 3 or 4 (Table 1). All event data goes through the Event Dispatcher which looks up for the ASL of a given event in the CSB table and builds a secure packet accordingly. Furthermore, when this event is received by the subscribing node it goes straight to the subscribing node's Event

Interceptor which looks up for the respective ASL and depending on the ASL values performs as follows:

- ASL-0 : since there is no security applied, forwards the Event_Data to the Subscribing Component.
- ASL-1: extracts the Event_Data and 1-way hash from the packet and forwards them to the Hash Engine which verifies the hash. In our evaluation we have used SHA-1 as the 1-way Hash. Once the hash is verified, Event_Data is forwarded to the Subscribing Component.
- ASL-2: extracts the Event_Data and Hash based Message Authentication Code from the packet and forwards them to the HMAC Engine which verifies the HMAC with the shared symmetric AES Key [11] between Publishing and Subscribing Node. Once the HMAC is verified, Event_ Data is forwarded to the Subscribing Component.
- ASL-3: extracts the AES cipher, forwards it to the Encrypt/Decrypt Engine which returns the plaintext i.e. Event_Data which is eventually forwarded to the Subscribing Component.
- ASL-4: extracts the SHA-1 hash and the AES cipher. Forwards the SHA-1 hash to the Hash Engine which verifies the hash. Once verification is successful, the AES cipher is fed to the Encrypt/Decrypt Engine which returns the plaintext i.e. Event_Data which is eventually forwarded to the Subscribing Component.

We have investigated the performance of DiFiSec communication with varied ASL levels with respect to full-fledged static security as well as no-security implementations. For a given set of nodes, the combined overhead λ^{Total} incurred by all the corresponding DiFiSec wirings in a network can be computed as in Eq. 1.

$$\lambda^{Total} = \sum_{w=0}^{n-1} \left\{ \left(\lambda_{i-asl}(\alpha_w) + \hat{\lambda}_{i-asl}(\beta_w) \right) \times {}^{\tau_w}/_{\mathcal{F}e_w} \right\} \quad (1)$$

where

n: Number of wirings

i-asl: Component Security Binding (CSB) level of a given wire

$\lambda_{i-asl}(\alpha_w)$: Event Dispatcher latency overhead for constructing data α_w for wire *'w'* with security level *i-asl* [$Resultant\ Packet \Longrightarrow \beta_w$]

$\hat{\lambda}_{i-asl}(\beta_w)$: Event Interceptor latency overhead to process packet β_w received with security level *i-asl*

$\mathcal{F}e_w$: Sampling period of event e_w

τ_w: Time interval when the event wiring *'w'* was active

In our implementation we deployed LooCI components on nodes having shared symmetric AES keys [11]. Component PubComp_A (ID-3) of size 2964 Bytes is deployed as Publisher Component whereas Component SubComp_B of size 2360 Bytes is deployed as Subscriber Component. Component PubComp_A is a LooCI component that senses the room temperature every 500ms and publishes it to the Distribution Channel. Network Gateway node performs the wiring through Telnet using DiFiSecWireTo and DiFiSecWireFrom calls and specifies a particular starting ASL level. Once wired, the Subscribing Node begins receiving temperature data with the specified security level.

As DiFiSec supports runtime adaptation; during the runtime the Gateway Node, the Publishing or the subscribing node can request a change in ASL for any given wiring. Gateway Node tweaks the security level of a wiring by calling aslAdapt specifying the Event Type, Source Component's ID, Source Node's MAC, Destination Component's ID, Destination Node's MAC and finally the desired Application Security Level. Consequently CSB entry gets updated and the very next event data when passes through the Event Dispatcher gets the packet constructed accordingly. Figure 4 and 5 show the latency overhead incurred as a result of sending and receiving Event_Data of a given size with varied ASL. In case of 2 Bytes the publishing component senses the Room Temperature and sends it to the subscribing component. As we can see the overhead incurred at Event Dispatcher and Intercepter increases considerably as the security level increases. We restricted our event data specimen upto 128 Bytes which we believe is sufficient for nearly all kinds of periodically sensed event data. We observe that overhead for preserving integrity $\boldsymbol{\lambda_{1-asl}}$ and $\hat{\boldsymbol{\lambda}}_{\boldsymbol{1-asl}}$ within this range of data makes no difference in terms of latency. We did however check integrity overhead for larger event data too, discovering 83ms for $\boldsymbol{\lambda_{1-asl}}$ and 81ms for $\hat{\boldsymbol{\lambda}}_{\boldsymbol{1-asl}}$ where event data size was 2560 Bytes. Other ASL levels however show a linear increase in latency as the size of the event data increases.

Event_Data	Event Dispatcher Latency (λ_{i-asl}) milliseconds			
Size(Bytes)	λ_{1-asl}	λ_{2-asl}	λ_{3-asl}	λ_{4-asl}
2 Bytes	54	453	660	723
8 Bytes	54	459	671	744
16 Bytes	54	461	684	764
32 Bytes	54	463	695	789
64 Bytes	54	471	709	812
128 Bytes	54	489	736	849

Figure 4 –Latency overhead (ms) at publishing node's Event Dispatcher

Event_Data	Event Interceptor Latency ($\hat{\lambda}_{i-asl}$) milliseconds			
Size(Bytes)	$\hat{\lambda}_{1-asl}$	$\hat{\lambda}_{2-asl}$	$\hat{\lambda}_{3-asl}$	$\hat{\lambda}_{4-asl}$
2 Bytes	54	472	733	813
8 Bytes	54	479	742	839
16 Bytes	54	486	753	871
32 Bytes	54	495	710	911
64 Bytes	54	511	731	939
128 Bytes	54	533	817	982

Figure 5 – Latency overhead (ms) at subscribing node's Event Interceptor

In order to see the overall benefit to the network, we followed Fig.1 scenario and compared its latency difference versus the same network implemented with full-fledge security (ASL-4). To keep it simple all the publishing components are sending 16 Bytes of data at a 30 seconds periodicity ($\mathcal{F}e_w = 30$sec) for 1 hour ($\tau_w = 3600$sec). For each run of 16 Bytes event data dissemination $\boldsymbol{\lambda_{1-asl}}$ takes

54ms, $\boldsymbol{\lambda_{2-asl}}$ takes 461ms and so on. The latency savings λ_{Δ}, therefore is the difference between security overheads incurred by full-fledged security and DiFiSec's tailored security provisioning:

$$\lambda_{\Delta} = \sum_{w=0}^{n-1} \left\{ \left(\lambda_{4-asl}(\alpha_w) + \hat{\lambda}_{4-asl}(\beta_w) \right) \times {}^{\tau_w}/_{\mathcal{F}e_w} \right\} - \sum_{w=0}^{n-1} \left\{ \left(\lambda_{i-asl}(\alpha_w) + \hat{\lambda}_{i-asl}(\beta_w) \right) \times {}^{\tau_w}/_{\mathcal{F}e_w} \right\} \quad (2)$$

We thus calculate the latency savings using DiFiSec for a period of 1 hour to be (1598.4 – 707.12) = 891.28 seconds, which is a considerable gain. These results also outline the significant variance distinct security operations have as well as their manifold impact in an event-driven environment.

VII. Conclusions And Future Work

Existing sensor network security schemes strive to achieve full-fledged but static security systems. In this paper though we made the case for security framework that facilitates dynamic and fine-grained security where applications are equipped to choose distinct pluggable security levels to be selectively performed on sets of communication solely on merit. DiFiSec offers 5 distinct levels of security which through Component Security Binding can be enacted at a fine-grained level of event communications between components. By employing DiFiSec's ASL Adaptation, applications can adjust these security bindings on runtime as per the changing network and system requirements. The feasibility of this approach has been demonstrated through a prototype implementation using the LooCI component model on the SunSPOT platform, and evaluation has shown that our approach is sufficiently lightweight to be applied effectively in WSN environments.

Our future work will focus upon the following key extensions (a) realizing location aware conditional Component Security Bindings through DiFiSec e.g. a Component Security Binding entry dictates only to be applied when either the sending or receiving components is outside the company's safe premises. (b) using DiFiSec to demonstrate varied security provisioning at the levels of selective events, interfaces and even receptacles e.g. a temperature component is normally set to maintain integrity for all temperature events but once the temperature rises beyond alarming threshold, it is sent with confidentiality, authentication and integrity. These features can only be achieved once a system offers such adaptability, varied security levels and fine-grained communication as DiFiSec.

Acknowledgment

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and NESSoS (Network of Excellence on Engineering Secure Future Internet Software Services and Systems) – the FP7 co-funded initiative to increase the trustworthiness of the Future Internet by improving the overall security of software services and systems

References

[1] Zhu, S., Setia, S., Jajodia, S.: LEAP: efficient security mechanisms for large-scale distributed sensor networks. Proceedings of the 10th ACM conference on Computer and communications security. ACM, Washington D.C., USA (2003).

[2] Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V. and Culler, D (2002) 'SPINS: Security Protocols for Sensor Networks', Wireless Networks, 8(5), 521-534.

[3] N. Matthys, S. R. Afzal, C. Huygens, D. Hughes, S. Michiels, W. Joosen, "Towards fine-grained and application-centric access control for wireless sensor networks", in proc. of the 2010 ACM Symposium on Applied Computing (SAC 2010)

[4] Perrig, A., Canetti, R., Tygar, J.D. and Song, D. (2002) 'The TESLA Broadcast Authentication Protocol', CryptoBytes, 5(2), 2-13.

[5] Costa P., et al., The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario, in proc. of the 5th Annual IEEE International Conference on Pervasive Computing and Comm. (PERCOM'07), New York, March 2007.

[6] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: A probabilistic approach. In 11th IEEE Inter'l Conference on Network Protocols (ICNP'03)

[7] T. Park and K.G. Shin, "LiSP: A Lightweight Security Protocol for Wireless Sensor Networks," ACM Trans. Embedded Computing Systems, vol. 3, no. 3, pp. 634-660, Aug. 2004.

[8] Coulson G., Blair G., Grace P., Taiani F., Joolia A., Lee K., Ueyama J. and Sivaharan T, A Generic Component Model for Building Systems Software, in ACM Transactions on Computer Systems, Vol. 26, No. 1, Feb 2008

[9] Sun Microsystems, Sun SPOT Sensor System, available online at http://research.sun.com/spotlight/SunSPOTSJune30 .pdf (accessed in September 2010)

[10] D. Hughes et al., "LooCI: A Loosely-Coupled Component Infrastructure for Networked Embedded Systems," Proc. 7th Int'l Conf. Mobile Computing and Multimedia Conf. (MoMM 09), ACM Press, 2009, pp. 195–20.

[11] J. Schaad and R. Housley. Advanced Encryption Standard (AES) key wrap algorithm. The Internet Engineering Task Force (IETF), RFC 3394, September 2002.

[12] SunSPOT Documentation. URL: http://sunspotworld.com/docs/index.html (accessed in September 2010).

[13] Afzal, S.R., Biswas, S., Koh, J.-b., Raza, T., Lee, G., Kim, D.-k.: RSRP: A Robust Secure Routing Protocol for Mobile Ad hoc Networks. In: Proc. of IEEE Conference on Wireless Communication and Networking, Las Vegas, NV, March 31- April 3, pp. 2313–2318 (2008), doi:10.1109/WCNC.2008.408

[14] Karlof, C., Sastry, N., Wagner, D. (2004) 'TinySec: A Link Layer Security Architecture for Wireless Sensor Networks', Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD, USA

[15] Sastry N., Wagner D., Security Considerations for IEEE 802.15.4 Networks, in the proc. of the 3rd ACM workshop on Wireless security (WiSe'04), Philadelphia, PA, USA, 2004, pp. 32 -42.

[16] H. Wang and Q. Li. Distributed user access control in sensor networks. In IEEE DCOSS International Conference on Distributed Computing in Sensor Systems, San Francisco, CA, June 2006.

[17] L. B. Oliveira, A. Kansal, B. Priyantha, M. Goraczko and F. Zhao, Secure-TWS: Authenticating node to multi-user communication in shared sensor networks, in "Proceedings of International Conference on Information Processing in Sensor Networks (IPSN'09)

[18] A. Dunkels, B. Grönvall and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", in proc. Of 29th IEEE conference on Local Computer Networks (LCN'04), Tampa, FL, USA, pp. 455–462, Nov. 16th-18th 2004.

[19] J. Rellermeyer and G. Alonso, "Concierge: A Service Platform for Resource-Constrained Devices", in ACM SIGOPS Operating Systems Review, Vol. 41, No. 3, pp. 245–258, 2007.

[20] D. Simon, C. Cifuentes, D. Cleal, J. Daniels and D. White, "Java on the Bare Metal of Wireless Sensor Devices: the Squawk Java Virtual Machine", in proc. of the 2nd International Conference on Virtual Execution Environments, Ottawa, Canada, pp. 78–88, 2006.