2012 Seventh International Conference on Availability, Reliability and Security

Topology-Driven Secure Initialization in Wireless Sensor Networks: A Tool-Assisted Approach

Stanislaus Stelle*, Mark Manulis*†, and Matthias Hollick‡
*Cryptographic Protocols Group, TU Darmstadt & CASED, Germany
†Department of Computing, University of Surrey, United Kingdom
‡Secure Mobile Networking Lab, TU Darmstadt & CASED, Germany
stanislaus.stelle@cased.de, mark@manulis.eu, matthias.hollick@seemoo.tu-darmstadt.de

***Abstract*—Secure initialization of sensor nodes with cryptographic keys is inherent to all security protocols and applications in the area of wireless sensor networks (WSN).**

We introduce a general framework, denoted `TOPKEY`, that provides tool assistance and performs secure initialization of sensor nodes with cryptographic keys over the air by leveraging the transmission power to confine the area in which potential attackers can eavesdrop on communication. Our analysis shows that physical protection based on transmission power may, in practice, lead to an acceptable level of key deployment security. Besides the fully automated key deployment, `TOPKEY` supports a five-step initialization process, suited to off-the-shelf sensor nodes that come without any pre-installed operating system. `TOPKEY` is currently tailored to static WSN topologies: it supports topology design and deploys topology-driven key generation for a range of WSN communication patterns.

We implemented the framework and analyzed its performance and scalability for commodity TelosB nodes and Contiki OS. Our analysis, performed with respect to different WSN topologies, shows that `TOPKEY` can be used to securely initialize a static network of about 100 nodes in less than one minute.

I. Introduction

A Wireless Sensor Network (WSN) is formed by sensor nodes that communicate with each other over the air in a multi-hop fashion. In typical WSN topologies nodes maintain wireless communication links with their direct neighbors and there exists some dedicated device (often referred to as sink) for administrative and application-specific purposes.

Whether WSNs should be used for measuring, processing, and transmitting of sensitive information depends on the amount of protection that can be enforced by its security mechanisms. WSNs have a range of applications and throughout network's lifetime applications might change, thus altering the security goals. It is well understood that security mechanisms cannot be enforced without taking care of cryptographic keys. Secure initialization of WSN nodes with cryptographic keys takes place in the initial deployment phase to allow for message authentication and/or encryption while bootstrapping the network. Secure initialization is inherent to many security protocols, e.g. secure routing [2], [17], [30], [41], authenticated in-network data processing [10], [15], [26], [33], possibly with further confidentiality support [6], [8], [31], [40], secure distributed data storage [14], or code updates performed over the air [20], [38], [39]. The assumption on secure initialization is crucial for their security and the central question of our work is how to turn this assumption into a practical solution?

The initial key deployment in sensor nodes takes place prior to WSN's enrollment. In static networks, topology design takes often into account specifics of the environment such as the required coverage area, existence of obstacles that may affect wireless transmission, identification of critical paths, etc. An a priori topology design is not applicable to dynamic or mobile networks, where topologies change over the time. Common to both network types, however, is the initial generation and distribution of cryptographic keys. WSN key distribution schemes (cf. Section V) support different types of keys, aiming to protect a range of basic communication patterns (cf. Section IV). Irrespective of the topology, generated keys must be securely deployed into the nodes. The key deployment process must also be practical. We argue that generation of keys and their deployment must be assisted by tools; otherwise it is unlikely that initialization can be carried out with a reasonable amount of effort, especially for large networks with high density of communication paths. Tool-assistance, however, brings additional risks: from the usability point of view one would like to harvest the benefit of wireless communications and perform key deployment over the air, whereas openness of wireless channels may result in leakage of transmitted keys to the adversary. Since prior to key deployment step, nodes may not be in possession of any cryptographic keys—which is precisely the purpose of secure initialization—protection of wireless key distribution must be realized by other means.

A. Related Work

Specification of WSN security protocols and their analysis is often carried out under the assumption that secret keys exist. A recent initialization approach by Perković et al. [28], termed Group Authentication Protocol (GAP), involves communication over two channels—a wireless channel for communication with the nodes and a visible light channel (VLC). In GAP one of the nodes acts as coordinator and all nodes interact initially to establish a (short) group authentication string (GAS), which is visualized to the user by each node. The user is then required to perform manual acknowledgement, e.g., by pushing a button on each node. Assume that some computing device holds secret keys that must be securely distributed to the nodes. Using GAS nodes can communicate their locally generated

978-0-7695-4775-6/12 $26.00 © 2012 IEEE
DOI 10.1109/ARES.2012.36

public keys to the computing device, which can then use public key encryption to secure key transport over the air. Secure initialization performed via GAP or related initialization protocols with out-of-band authentication, e.g. [23], [29], [35]–[37], inherently requires public key cryptography, unless some physical protection mechanisms aiming to prevent interference with the attacker are in place. In this latter domain the most notable solution was proposed by Kuo et al. [18] — a Faraday cage was used as a physical shield to protect transmission of keys. In [18] one node is seen as a keying device and is placed together with other nodes into the Faraday cage to perform key distribution. The more nodes can fit into the cage, the more cost-effective this solution becomes. As argued in [7], a Faraday cage may not completely shield from interference attacks. However, experimental results in [18], demonstrate that the amount of achievable protection is sufficient for practical purposes. As noticed in [28], such cage introduces an additional overhead to the user and requires further protection mechanisms to ensure that the keying process continues only if the Faraday cage is closed. Is it possible to perform secure initialization process independently of any external devices, while still being able to deploy keys over the air with a practically acceptable level of protection? Anderson et al. [4] deploy cryptographic keys in plain over the air if communication is required hence rendering the communication vulnerable only during this phase. However, the transmission of plaintext keys can occur after deployment and is therefor not observeable by responsible entities and interceptable by adversaries. Our work follows this line of research and explores another approach for automated deployment of cryptographic keys in WSNs.

II. Overview of Our Contributions

We propose a tool-assisted approach for secure initialization of WSN nodes with cryptographic keys to protect basic communication patterns. We develop a framework, called TOPKEY which is currently tailored to static WSNs and thus supports the topology design. The initialization approach is nonetheless modular and can be adopted to the needs of dynamic networks. Secure key deployment is an important part of the framework, which however provides automated support for a more general five-step initialization process, tailored to off-the-shelf nodes:

Node flashing Sensor nodes are flashed with the extended operating system (OS) image to provide software support for their communication with the TOPKEY tool. This step is necessary as off-the-shelf nodes are typically delivered without any pre-installed OS.

Node registration Sensor nodes are initially registered with TOPKEY that builds a centralized view on all the WSN nodes. TOPKEY offers several ways to perform the registration, aiming to increase its usability.

Topology design Registered nodes can be used to design static WSN topology by defining single-hop connectivity links amongst the nodes and towards the base station. TOPKEY offers automated support for topology design and can verify its soundness. It comes with a GUI to increase the usability. (This step could be omitted in case of dynamic WSN topologies.)

Key Generation TOPKEY performs then automatic generation of cryptographic keys for the later protection of basic communication patterns amongst the WSN nodes and the base station. Keys are currently generated in a deterministic approach, i.e. their total number and types are determined by the topology. Such topology-driven approach is best suited for static networks, where connectivity of nodes and frequent communication patterns do not change over the time. (TOPKEY can be extended with other key generation algorithms to enable secure initialization for dynamic networks.)

Key Deployment In this final initialization stage generated keys are securely deployed into sensor nodes. TOPKEY can perform key deployment in several secure ways. The most usable approach, yet also most challenging from the security point of view, is the distribution over the air. This approach is realized in TOPKEY by reducing the *transmission power* with which keys are distributed. Regulating the transmission power is our way to provide physical protection against eavesdroppers. We observe that without external physical devices, such as Faraday cage [18], further assumptions must be in place, e.g. that attackers cannot eavesdrop on the communication beyond a certain range. In order to understand limits of this approach and to illustrate its behavior in practice we perform experiments and determine acceptable ranges in which transmission of keys over the air remains secure and reliable.

The modularity of TOPKEY implies benefits for usability: for example, different steps of the initialization process can be performed by distinct parties and on different computing devices. While security-critical steps such as key generation and deployment are likely to be performed on the same device, the initial steps for node flashing, node registration, and topology design can be outsourced to other devices under different administrative control.

Our performance analysis of TOPKEY has a theoretical part, where we estimate the required communication complexity of the topology-driven key distribution approach based on network size and connectivity, and an experimental part, where real measurements with TelosB compatible motes [32] running Contiki OS [12] were performed to obtain timings of key distribution for different network topologies. We released the developed TOPKEY tool into the open-source domain [1].

III. WSN Topologies

Definition 1 (WSN Topology): A *WSN topology* is given by a graph $\Gamma = (V, E)$, where $V = \{v_i\}_i$ denotes the set of sensor nodes and $E = \{e_{i,j} | v_i, v_j \in V\}$ is the set of single-hop links amongst the network nodes. Each element $e_{i,j} \in E$ determines a pair of *neighbors* (v_i, v_j). For each node $v_i \in V$ we further define a *cluster* comprising all neighbors $v_j \in V$, i.e. $e_{i,j} \in E$.

A WSN topology is *static* if the set of nodes and their connectivity does not change over time; otherwise the topology is *dynamic* (c.f. [34] for a more general classification). Our main focus is on static WSN topologies that are frequently installed in pre-defined indoor and outdoor areas to trigger fire alarms or warn against trespassing, in the industrial automation

to monitor the manufacturing process conditions, or in civil engineering to warn about deviations and material degradation. These applications often require a careful topology design to be performed in advance, which contrasts dynamic topologies that mostly arise in a mobile environment, e.g. in vehicular communications.

Let $|V|$ denote the number of nodes (*network size*) and $|E|$ the number of single-hop links. For any WSN topology there is an upper bound $|E| \leq \frac{|V|(|V|-1)}{2}$, which corresponds to a complete graph. However, in practice not all pairs of nodes will become neighbors. We thus recall some basic topologies and indicate upper bounds to give intuition about the growth of $|E|$ in practice.

Linear topology All nodes of the network can be connected via a multi-hop path such that each node is visited at most once. This topology has the upper bound $|E| \leq |V| - 1$.

Tree topology In an n-ary tree topology sensor nodes form a tree with each node having at most n child nodes. The upper bound in this topology is also $|E| \leq |V| - 1$.

Grid topology In a grid topology each sensor node is connected to at most two other nodes along each dimension. A two-dimensional grid topology, i.e. $|V| = n \cdot m$, has the upper bound $|E| \leq n(m-1) + (n-1)m \leq 2|V| - 2\sqrt{|V|}$.

IV. WSN Communication Patterns

Typical WSN traffic can be characterized according to several, more or less frequent, basic communication patterns:

Sink-to-Node(s) This pattern assumes that the sink sends its message to one or more nodes, via a network-wide broadcast. In case the transmission power of the sink is high enough, the message can reach all nodes in one hop; otherwise delivery of messages to distant nodes is performed by routing mechanisms to ensure that messages are propagated across the network. The communication pattern in which sink's message is sent to all nodes is very frequent and is often used for administrative purposes, e.g. to query WSN or to update the code.

Node-to-Sink Another basic pattern occurs when some node $v_i \in V$ wishes to send a message to the sink device. Delivery of this message may require traversal along a multi-hop path, typically determined through applied routing protocols. This pattern is very frequent in WSNs that are primarily used for monitoring and reporting of the environmental data.

Neighbor-to-Neighbor(s) This pattern applies when a node $v_i \in V$ wishes to send a message to its neighbor v_j or to all of its neighbors (cluster) $\{v_j\}_j$ with $e_{i,j} \in E$. Every sensor node is a transceiver. Due to the broadcast nature of the signal any message sent by v_i is delivered to all nodes in its cluster. Therefore, targeted messages to specific neighbors must be addressed appropriately.

Node-to-Node This pattern occurs when two (non-neighboring) nodes $v_i, v_j \in V$ with $e_{i,j} \notin E$ wish to exchange messages that are then typically forwarded via routing protocols along the multi-hop path connecting the two nodes.

The type and frequency of utilization of the above patterns depends on the applications as well as on the WSN topology.

V. Key Distribution in WSN

In this section we review related work on key distribution in WSNs. We then motivate and describe the topology-driven approach for which our `TOPKEY` framework currently provides tool-assistance. The focus is laid on cryptographic keys that are used with symmetric primitives, such as block ciphers and message authentication codes, for which hardware support is often provided by existing sensor nodes and which are in general better suited for resource-constraint devices.

A. Keys for Different Patterns

In order to protect WSN communication patterns using symmetric cryptography it is necessary for a node to have independent keys of different types that it shares with other entities (nodes/sink) within the same communication pattern. In particular, each node may need an *individual key*, shared with the base station and *pairwise keys* to protect pairwise communication with other nodes, i.e. either using direct communication between neighbors (link key) or indirect communication over a multi-hop path (path key). It may also need a *group key* shared with all WSN nodes and possibly with the base station to protect network-wide broadcast messages and *cluster keys* to alow secure communication amongst some subset of nodes; a typical cluster in static WSN topologies comprises all neighbors of a particular node.

B. Key Distribution Mechanisms

Many existing WSN key distribution schemes provide nodes with some preliminary key material that—once the network starts operating—is used by nodes to dynamically compute symmetric keys for different patterns. Sensors perform this computation typically through interaction with each other (mostly for computing pairwise keys), but without any online involvement of the party that performed the initialization step. This approach is best suited for ad-hoc type of networks with dynamic topologies or, more generally, for WSN topologies that are *not* known prior to the operative deployment of nodes. Therefore, we call this form of key distribution **topology-unaware**. Its main strength is the establishment of pairwise keys between any pair of nodes, i.e. in dynamic topologies where neighbor relationships are not fixed. We recall and provide a concise comparison of topology-unaware key distribution mechanisms in Table I, according to the following criteria:

- Can the approach guarantee that for each supported key type all nodes in the same pattern will indeed be able to compute the same key? This is the case for deterministic mechanisms, whereas probabilistic schemes may not always provide nodes with the desired keys.
- The amount of pre-deployed keying material and different key types that can be dynamically computed by nodes. Table I indicates whether mechanisms support computation of a group key (gk), individual keys (ik), link keys (lk) for protecting communication amongst neighbors, path keys (pk) to allow secure communication amongst non-neighbors, and cluster keys (ck). Existing

TABLE I
BRIEF COMPARISON OF WSN KEY DISTRIBUTION MECHANISMS

key distribution mechanism	deployment guarantee	key types		complexity	
		pre-deployed	all supported	computation	communication
LEAP [43], [44]	deterministic	ik, gk	ik, lk, pk, ck, gk	low	low
BROSK [19]	deterministic	gk	lk, gk	low	low
IOS [21]	deterministic	ik, lk	ik, lk	medium	medium
DMBS [21]	deterministic	ik, aux	ik, lk	medium	medium
Merkle's puzzle [27]	deterministic	-	ik	high	high
random key assignments [13]	probabilistic	aux	lk, pk	medium	medium
via transmission range [16]	probabilistic	aux	lk, pk	medium	medium
q-composite approach [9]	probabilistic	aux	lk, pk	medium	medium
polynomial approach [5]	deterministic	ik, aux	ik, lk, pk	medium	medium
polynomial approach [25]	probabilistic	ik, aux	ik, lk, pk	medium	medium
Blom-based [11]	probabilistic	ik, aux	ik, lk, pk	medium	medium
with deployment knowledge [42]	probabilistic	aux	lk, pk	medium	medium
location-based [24]	probabilistic	ik, lk	ik, lk, pk	medium	medium

mechanisms mostly require that certain types of keys are pre-deployed, possibly with some auxiliary information (aux), in order to allow dynamic derivation of keys.
- The required costs for computation and communication, primarily on the side of the sensor nodes.

According to Table I, distribution performed with LEAP [43], [44] offers best diversity of supported key types and patterns. The majority of schemes, however, aim to establish shared keys amongst any pair of (non-neighboring) nodes and is thus best suited to dynamic networks.

Our focus is on static WSN topologies that are designed prior to the installation of the network. This setting offers a number of benefits for key distribution. Since neighbor relationships are fixed, the knowledge of the topology can simplify the generation of independent keys for the most frequent communication patterns: sink-to-node(s), node-to-sink, and neighbor-to-neighbor(s). We consider this approach for key distribution as being **topology-driven**. Unlike the topology-unaware schemes that could also be used for the purpose of key distribution in static, predefined WSN topologies, the topology-driven approach reliefs sensors from communication and computation overhead, while still taking the initialization party off-line once the network starts to operate.

VI. TOPKEY: TOPOLOGY-DRIVEN KEY DISTRIBUTION FRAMEWORK

We describe our framework TOPKEY for topology-driven WSN initialization and key distribution. TOPKEY offers automated support for a five-step secure initialization process and tool-assistance for topology design, generation of cryptographic keys, and their secure deployment. We elaborate on each of the five steps, mentioning the expected outcome and discussing challenges from the security, usability, and technology point of view.

A. Setup Environment and Technology

TOPKEY operates in a setting, where some computing device (*setup device*) is used to design WSN topology and to securely initialize its nodes. In order to realize most of the steps over the air, one of the sensor nodes is connected to the setup device and acts as a *gateway* for communication with other nodes. We assume that all nodes come initially without any pre-installed software, as is often the case with off-the-shelf nodes, which often lack an OS.

1) The Tool: Importance of tools in secure sensor node initialization is obvious: with the increasing size of the network only tool-assisted initialization can offer the desired level of scalability by means of automation. Another important aspect is the usability of the tool-assisted approach. We have implemented our TOPKEY framework as a fully working prototype. Its current logic is implemented and tested in Java 1.6 to achieve sufficient level of platform independence. TOPKEY tool comes with its own GUI to assist the user throughout different steps of the initialization process. This GUI and the processing of user inputs is realized using the JGraph [3] library with minor modifications to enhance the usability of the framework.

2) Sensor Nodes: TOPKEY is currently tailored to Crossbow's TelosB (or compatible) nodes. TelosB notes are among the most popular sensor nodes and widely used in the scientific community for experimental purposes; they are relatively cheap and available off-the-shelf. Their hardware specification includes an MSP430 platform with an 8MHz processor and 10kB of RAM. The non-volatile memory amounts to 48kB. Communication amongst the nodes is realized using IEEE 802.15.4. TelosB nodes can be powered with two AA batteries or over an USB port. In terms of sensing abilities, TelosB nodes can sense light, humidity, and temperature. Our initialization approach is application agnostic and is not limited with respect to the provided sensing abilities.

TelosB nodes can work with different operating systems. TOPKEY is currently based on the open-source Contiki OS [12], in version[1] 2.5—a highly portable, multi-tasking operating system with fast and event-driven architecture, which is very well suited for memory-efficient networked embedded systems. In this way distributed keys can persistently be stored on the nodes, despite of possible battery exchange. Each node

[1]Contiki OS is currently under active development of the community, with its latest version being released just few months ago.

running Contiki OS is equipped with a micro IP which depends on the manufacturing process and is assigned to the node automatically after flashing. A micro IP has two 8-bit blocks (e.g. 127.12).

Contiki OS has some benefits over other OS for sensor nodes: for example, it is less overloaded than say TinyOS [22] so that images become compact and suit better the constraints of the restricted memory in TelosB nodes. Contiki OS is also easier to program for due to the use of the native C (rather than the specialized nesC of TinyOS). TOPKEY can nonetheless be extended to function with other OS since many automated steps of the initialization process are OS-independent.

3) Setup Device: The setup device executes TOPKEY. Its implementation in Java admits usage of any commodity computing device with a screen (e.g. desktop, laptop, or tablet PC). We do not assume that setup device supports direct communication with sensor nodes based on IEEE 802.15.4 — in fact, this standard is currently not supported by the majority of computing devices. This is why our framework currently utilizes one of the sensor nodes as a gateway that will be connected to the setup device through the USB port.

B. Tool-Assisted Secure Initialization Process

The proposed tool-assisted secure initialization process is given by a series of basic initialization steps (as introduced in Section II): node flashing, node registration, topology design, key generation, and key deployment. These steps have to be performed in the mentioned order due to their dependencies. TOPKEY offers convenient user interaction and fully-automated support for the last three steps, whereas the node flashing and node registration require user involvement to enable the actual over-the-air communication between the setup device and the nodes. This involvement is required since nodes come without OS and their over-the-air connectivity with the setup device (through a gateway node) has to be established in the first place. This effort is comparable to VLC [28]. The proposed initialization process is highly modular, which results in several benefits as also highlighted in the following.

1) Node Flashing: In *node flashing* step an OS image with software support for TOPKEY is deployed on each node. This is realized through the USB connection, established between the setup device and the sensor node by the user. Image deployment can be parallelized using USB hubs. The flashing process copies OS images to the nodes and executes a local installation procedure. TOPKEY supports node flashing by offering Contiki OS images equipped with the additional logic for tool assistance. The image size amounts to 39 kB (only 3 kB more than the native Contiki OS image including linked lists, network connections and the file system).

Nodes can be flashed by any party, not necessarily the one that will supervise later initialization steps. For instance, nodes can be flashed by manufacturers. Although node flashing doesn't involve any information related to the security part of the initialization process, care should be taken to ensure that installed images are malware-free. This might be the case if nodes are flashed by an untrusted party. Verifying whether an image is trustworthy can be realized using standard techniques, e.g. by measuring and comparing its cryptographic hash value, if the underlying platform supports this.

2) Node Registration: Node registration step makes the tool aware of nodes that will form future WSN topology. In particular, registration must ensure that micro IPs of the sensor nodes become known within TOPKEY tool for later over-the-air communication. Our TOPKEY framework offers several ways for node registration.

The first approach is manual, yet still user friendly: TOPKEY GUI adds new nodes via a double click or the menu option. The editing of the micro IP is then performed manually by the user. Obviously, the user must know the micro IP for each registered node. This approach is usable for a rather small sensor network. TOPKEY tool can speed it up by offering support for batch registration, using a list of all micro IPs.

The second approach is semi-automated in that it reliefs the user from manually adding and editing the micro IPs of sensor nodes or collecting this information manually beforehand. This is realized by requiring the user to plug in TelosB nodes into the USB port of the setup device. The actual registration of nodes in the TOPKEY framework is then performed automatically by reading out the micro IPs of the attached nodes and displaying the registered nodes on the GUI. This approach scales better as the only manual action of the user is to plug and unplug the nodes into the USB port of the setup device. This semi-automated approach is possible since flashed nodes already contain software support for the communication with TOPKEY tool and the framework provides platform-dependent driver for the setup device to communicate with nodes via the USB connection.

Observe that a fully automated node registration would require that nodes continuously send out their registration information that must then be recognized and processed by the initialization tool. Any on-demand request of this information would introduce user's manual involvement. Such fully automated approach, however, means that a continuous power supply for sensor nodes must be in place between the node flashing and the registration steps, which may not be practical in most cases.

At the end, registered nodes and their micro IPs become visible on TOPKEY GUI. The tool automatically verifies that micro IPs of registered nodes have a correct format. The user can thus proceed with the topology design. The usability of node registration is further enhanced by the ability to store registration information in an XML format and to load it later for further processing. This introduces modularity that separates node registration from topology design.

3) Topology Design: This step must take into account the multi-hop nature of WSNs. TOPKEY allows the user to design topologies using the available GUI: the user can arrange nodes and draw two types of connections—(a) between the sink and any other node that is supposed to be reachable in a single hop from the sink, and (b) between any pair of nodes that are supposed to become neighbors. In this way, for every node a cluster comprising all its neighbors can be automatically

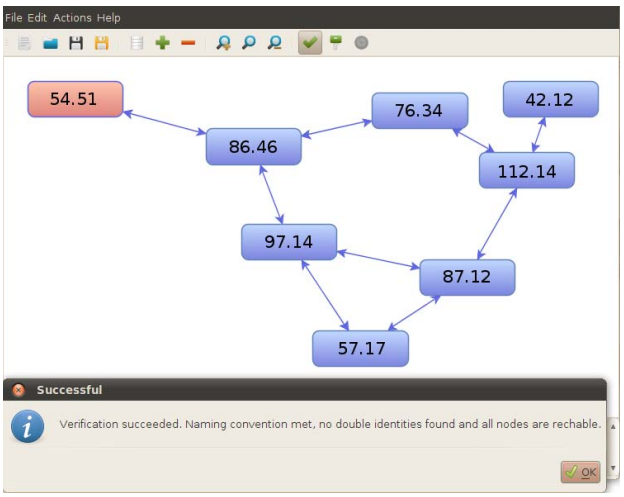

Fig. 1. Topology design in TOPKEY

detected within the tool. For convenience, TOPKEY offers XML-based storage and import of the designed topology.

The second important step in the topology design is to verify soundness of the topology, according to several criteria: first criterion is that all nodes must be reachable. This means that every sensor node should be able to receive messages sent by the sink either directly in a single hop or indirectly through a multi-hop path via other nodes. Second criterion is the uniqueness of micro IPs, as those are used by Contiki OS for addressing purposes. TOPKEY offers fully automated support for topology verification and reports identified problems (if any) to the user. Any sound topology is ready to serve as a basis for the key generation step. Figure 1 shows an example of a sound topology designed with TOPKEY.

Topology design is another modular step that can be performed even if sensor nodes are not physically available to the designer, i.e. by a third party, and the designed topology can be loaded into the setup device prior to the next step.

4) Topology-Driven Key Generation: The first security-critical initialization step is the generation of cryptographic keys. It is assumed that a sound WSN topology graph $\Gamma = (V, E)$ is available within the framework. The goal of the *key generation* step is to produce cryptographic keys for the most frequent communication patterns. Based on our discussion in Section III and the analysis in Section V the highest level of flexibility with regard to different key types is offered by LEAP [43]. The knowledge of the topology graph Γ can thereby significantly simplify underlying key generation mechanism, as all keys can be computed centrally on the setup device, rather than locally on the sensor nodes. This eliminates the need for additional computations on the node side as discussed in Section V.

TOPKEY generates four types of (symmetric) keys:

- a *group key* that can be shared between the sink and all nodes in V and used to protect the sink-to-nodes pattern,
- an *individual key* for each node $v_i \in V$ that can be shared between v_i and the sink and offer protection for the sink-to-node and node-to-sink patterns,
- an *individual cluster key* for each node $v_i \in V$ that can be shared between v_i and all $v_j \in V$ with $e_{i,j} \in E$, and used to protect a node-to-neighbors pattern, and
- a *pairwise key* for each pair of neighbors v_i, v_j with $e_{i,j} \in E$ that can be used to protect their neighbor-to-neighbor communication.

All generated keys are independent due to the use of fresh randomness within the Java Security Extension. TOPKEY allows for parameterizing lengths of these keys, however, the default length is set to 128 bits, e.g. for AES-128 block cipher that is supported by TelosB hardware.

TOPKEY saves generated keys in a local structure. If required, keys can be stored to the hard disk of the setup device in an XML format and loaded later for redeployment or modification due to possible updates of the network topology, or if certain keys have to be generated anew. In general, we assume that the setup device, performing the generation of keys, is trustworthy, as it would not be possible to securely initialize off-the-shelf nodes without making such assumption on the key generation procedure. Once cryptographic keys are generated the user can proceed with their deployment, which is the final step of the initialization procedure.

5) Key Deployment: Availability of a sound WSN topology $\Gamma = (V, E)$ and a set of generated topology-based cryptographic keys K are the prerequisites of the *key deployment* step, whose goal is to transmit keys from K onto the corresponding sensor nodes in V. We are facing two main challenges with regard to the deployment procedure as discussed in the following.

The first challenge is to distribute keys securely, i.e. inaccessible to an attacker. The security problem is apparent in the fact that nodes at this stage are not equipped with any cryptographic keys. One may think that deploying a master secret key into each node during node flashing or node registration would ease the problem. However, this is not the case as it just shifts part of the problem and introduces unnecessary security risks to the those initialization steps that are currently less concerned with security and could be performed by third parties. The most secure way is to manually plug every node into the setup device and perform the deployment by transmitting the keys over the serial USB connection. Although this possibility is considered in TOPKEY, it is clearly not scalable in that the amount of user interaction increases linearly with the network size. This brings us to the second challenge—the usability of the distribution process. Aiming to minimize user interaction we opt for a fully automated approach: the crucial idea behind TOPKEY is to harness the benefit of a wireless channel and perform distribution over the air. The setup device, over a USB-connected gateway node, establishes wireless communication with the remaining nodes based on IEEE 802.15.4

Regulating Transmission Power. The automated process

for wireless key distribution described above introduces a technical challenge behind the TOPKEY key deployment procedure—nodes must remain within the range of the wireless signal to reliably receive cryptographic keys, while no attacker should be able to interfere with the communication; in particular, our main concern are eavesdropping attacks by which the attacker could try to sniff on the keys that are transmitted in plain. As a promising solution we consider reduction of the transmission power with which the gateway node is sending out the keys. A too weak transmission power, however, may increase the packet loss. This leads to another trade-off between security and usability and to definitions of two range types:

Secure range This is the minimal radius, measured from the location of the gateway node, at which the packet loss amounts to 100%. This means that over-the-air key distribution can be performed securely as long as no attacker is present within this range.

Reliable range This is the maximal radius, measured from the location of the gateway node, at which the packet loss amounts to 0%. This means that over-the-air key distribution can be performed reliably (with none of the keys being lost) as long as all sensor nodes in V are located within this range.

The central question is, how do these ranges behave with varying transmission power of the gateway node? In Table II we measure secure and reliable ranges in dependency of the transmission power through real-world experiments using TOPKEY and TelosB nodes. In our attacker model and experiments we do not assume that eavesdropper's abilities are superior (e.g. using powerful antennas) over those of commodity sensor nodes. The first column depicts the transmission power of a TelosB node. It can be set to an integer value, between 0 (min) and 31 (max). From the measurements we conclude that transmission powers between 3 and 7 seem to be of main interest for practical purposes. In particular, smaller transmission powers result in noticeable packet loss and far to small reliable ranges, whereas higher powers increase the range and by this the security risk. In addition to secure and reliable ranges we also measured the average round trip time (RTT) for a single packet communicated between the gateway node and a TelosB node located at the reliable range distance. For all measurements this RTT value was 15 ms. We used the nullmac driver instead of the native Contiki OS driver to avoid overhead at the MAC layer. RTT, however, will become important in the scalability analysis of the key deployment process (c.f. Section VII).

TABLE II
IMPACT OF THE TRANSMISSION POWER

Power	Secure range	Reliable range
3 (10%)	4m	0.5m
4 (13%)	4.5m	1m
5 (16%)	5m	1.5m
6 (19%)	6m	1.5m
7 (23%)	7.5m	2m

The above measurements were performed in a normal office environment with a free line-of-sight between the nodes. This also means that having obstacles would likely reduce the resulting ranges. Each measurement was replicated 100 times, with their average value given in Table II. The actual choice of suitable transmission power depends on the environment in which the initialization is performed. In general, we expect that initialization takes place in a closed office environment or at home, for which transmission powers between 3 and 5 would be suitable due to lower secure ranges, e.g. if the user cannot oversee the whole area within the secure range. Transmission power can be set slightly higher if the boundaries offer stronger absorption.

Optimized Key Packaging and Format. In addition to the reliable range, the usability of the key deployment step is influenced by the overall time that is required to distribute all keys from K to the nodes in V. This time is dominated by the number of transmitted packets and the packet loss. TOPKEY we developed an optimized way for the packaging of keys, depending on the length of each key and on the admissible packet size that is often determined by the TCP/IP protocol stack of the operating system. As our TOPKEY framework currently provides support for TelosB nodes with Contiki OS (using its IP protocol stack) we elaborate on the optimal packaging of keys with respect to this technology. The goal of optimization is to reduce the number of packets.

For (unreliable) unicast transmissions Contiki OS offers 108 byte for payload per each micro IP packet. The default length of a symmetric key from the TOPKEY key generation step is 128 bits (16 bytes) to suit the TelosB hardware support for the AES-128 block cipher. Some key types that our TOPKEY tool generates for a node v_i do not depend on the actual WSN topology, namely the group key, the individual key of v_i, and the cluster key of v_i. It appears reasonable to package those keys into a single unicast packet with the following format where the 8-bit header information is followed by the payload

8 bits	128 bits	128 bits	128 bits	16 bits
header	ik	gk	ck	crc

containing the individual key (ik), the group key (gk), and the cluster key (ck), each 128 bits long, together with the 16 bit cyclic redundancy code (crc) to allow detection of accidental changes on the receiver's side. The constant length of this packet amounts to 51 bytes and can be easily fitted into the standard Contiki OS' unicast packet.

In contrast, the amount of pairwise keys as well as the amount of other nodes' cluster keys that have to be sent to v_i depends on the number of its neighbors. We package those keys using a different format where $5 \cdot 19$ bytes for the payload

8 bits	$5 \cdot 19$ bytes	16 bits
header	payload	crc

part indicate that up to five entries (each 19 bytes long) can be sent in one packet. The maximal packet length is 98 bytes and can easily fit into one Contiki OS' (unreliable) unicast

packet. Each of the at most five entries per packet consists of one leading byte to indicate whether the entry represents a pairwise key or a cluster key. Out of 18 remaining bytes the first two bytes contain the micro IP address of the node v_j with which the receiver v_i should associate the received pairwise/cluster key. This micro IP will be used by Contiki OS to address nodes that are intended to receive the packet protected with the given key after network's roll-out. The last 16 bytes of each entry contain the actual cryptographic (pairwise or cluster) key. At first sight it appears as a waste to use a whole (leading) byte per entry to indicate the type of the associated key. However, due to the limited payload size of the Contiki OS' unicast packets there is no difference whether this information is encoded into one bit or one byte, as either way at most five entries (keys) would fit into one unicast packet. However, alignment to bytes makes it more efficient to reconstruct the keys and the micro IP addresses from within the Contiki OS. The indication of the key type (pairwise/cluster) per entry rather than per packet allows for a compact packaging of keys since every node v_i receives one pairwise key and one cluster key for each neighbor v_j. For example, assume that v_i has six neighbors resulting in six pairwise and six cluster keys. Since at most five keys can be transmitted in one packet the indication of the key type per packet would result in four packets (two for pairwise and two for cluster keys) whereas with our approach only three packets would be required in total.

Finally, we note that the *header* part in both packet formats consists of eight bits, from which the first two denote the packet type: if equal to '00' then the packet is assumed to carry ik, gk, and ck type of keys; and if equal to '01' then the receiver expects to receive pairwise and cluster keys of its neighbors. If, however, the packet type has a leading '1' then its second bit is ignored and the payload of the packet is interpreted as a command; such packet type '1x' is currently not used by the framework but is reserved for the future. The header bits at positions 2 to 4 are used to encode the sequence number of the packet — each `TOPKEY` communication session between the setup device and a sensor node v_i will start with the sequence number 0 so that up to eight sequentially numbered packets can be transmitted to v_i. The encoding of the sequence number with three bits gives the total capacity to transmit up to 35 pairwise and cluster keys per node, in addition to the group key, the individual key, and the own cluster key of that node. Due to one-to-one dependency between a pairwise key with a neighbor and that neighbors' cluster key this approach allows transmission of up to 17 pairwise and 17 cluster keys to each v_i, which is more than sufficient for practical WSN topologies, i.e. in practice we do not expect a node to have more than 3-4 neighbors. The final three bits of the header are used as a counter for the number of packets that are still to come. This information allows each node v_i to calculate how many packets should have been received so far (the sum of the sequence number and the counter) and decide whether any intermediate packet was lost. We will discuss the packet loss problem and reliability issues in the next paragraph.

Packet Transmission and Reliability. In the key transmission process, for each node in V the setup device starts by sending out packets of the first type, containing the three topology-independent keys (gk, ik, ck). Each of these packets is followed by a sequence of packets of the second type, containing the topology-dependent pairwise and cluster keys of the neighbors. Every transmitted packet to v_i contains the sequence number, the counter for the number of packets that are still to come, and the crc value, allowing each receiving node to check that no bits were accidentally flipped. Upon receiving a packet, nodes check the crc value and eventually parse the keys according to the described format. The keys are then stored in the non-volatile memory using the file system of Contiki OS.

Packet transmission is realized using Contiki OS' unreliable unicast due to the better payload/transmission delay ratio in comparison to its reliable version that acknowledges receipt of each packet by default. In `TOPKEY` framework we opted for the unreliable unicast since the distribution of keys should be performed to nodes that are assumed to be located within the reliable range. Even though our measurements in Table II indicate reliable ranges with 100% delivery, in practice the transmission of packets within this range may still result in successful delivery for most of the packets but not necessarily for all. Therefore, some reliability mechanisms must still be in place, should our framework remain usable in practice. Instead of using reliable unicast of Contiki OS we adopt a different approach: the setup device sets locally a timeout for each sensor node. Each node is expected to respond with an ACK message only after obtaining all (valid) packets. If the setup device doesn't receive an ACK message prior to the timeout then all packets for that node have to be repeated. At the same time, each node locally keeps track on the arriving packets and sets its own timeout after having received the first packet. The node keeps track on the occurring problems, e.g. if the crc check for some packet fails or a packet has not been delivered, and sends a NACK message containing sequence numbers of such packets to the setup device after its local timeout or after receiving the last packet. Since each sequence number is only 3 bits long there can only be 7 packets lost — meaning that an NACK packet can hold sequence numbers for all lost packets. The setup device then re-sends packets for which the node encountered problems and waits for an ACK of that node. Since key packets in `TOPKEY` are transmitted within the reliable range, sending a single ACK message at the end to acknowledge the correct delivery of all packets leads to better performance.

VII. Scalability and Performance

Here we are interested in the complexity of the automated support offered by our `TOPKEY` framework. Since generation of symmetric keys in K for a given topology $\Gamma = (V, E)$ is performed locally on the setup device its costs are negligible in comparison to the costs of key deployment that are dominated

by wireless communication between the setup device and the nodes. First, we consider the total number of keys (size of K) that must be computed and distributed using our topology-driven approach. Taking into account that during key deployment the gateway transmits keys over the air to each node individually, we are further interested in the total number of required single-hop transmissions. Lemma 1 shows that the size of K and the upper-bound on single-hop transmissions (assuming that each key per node requires one single-hop transmission) is determined solely by the number of nodes $|V|$ and the number of single-hop links $|E|$.

Lemma 1 (Keys and Single-Hop Transmissions): Let $\Gamma = (V, E)$ be a WSN topology and K the set of keys according to the key generation step from Section VI-B4. Then:

1) The total number of generated keys is $|K| = 2|V| + |E| + 1$.
2) The upper-bound of single-hop single-key transmissions in the key deployment step is $3|V| + 4|E|$.

Proof: The key generation procedure computes two keys (individual and cluster key) for each node $v_i \in V$, one pairwise key for each pair of neighbors v_i, v_j with $e_{i,j} \in E$, and a single group key, which gives the desired total number of keys. As for the number of single-hop transmissions observe that each node v_i expects to receive a group key, its individual key, and the cluster key, which gives us $3|V|$ single-key transmissions to accomplish this task. Additionally, for each single-hop link $e_{i,j} \in E$ the pairwise key between v_i and v_j must be sent to both v_i and v_j, the cluster key of v_i must be sent to v_j, and the cluster key of v_j must be sent to v_i, which results in additional $4|E|$ transmissions. ∎

In practice the key packaging applied in TOPKEY (cf. Section VI-B5) significantly reduces the amount of single-hop transmissions in the key deployment step. We now present concrete timings, measured with TelosB nodes for different types of WSN topologies from Section III. More precisely, we generated multiple random instances of different WSN topologies and measured the resulting key deployment time for networks of up to 20 TelosB nodes. The transmission power of the gateway node was set to 5 (16%) that according to Table II results in 1.5 m reliable range and has average RTT of 15 ms. The resulting key deployment timings are plotted in Figure 2. To keep the TelosB nodes from rebooting due to too fast transmissions we needed to add delays which increased the deployment by the factor of 41. These delays are not necessary if more stable nodes are used. Hence the following presentation accounts for the deployment time without the added delays.

For a network of 20 nodes TOPKEY takes less than 3 seconds, depending on the topology, to perform automatic key deployment. Timings for a complete graph-topology, however, should be interpreted as a feasibility result since practical topologies typically have sparser connectivity. Timings that seem more suitable for practical purposes reside in the interval given by measurements for linear/tree and grid topologies. In this case TOPKEY would require 13 seconds on average to perform key deployment for 20 nodes. We further apply our measurements to estimate performance scalability in larger networks. For example, assuming 15 ms average RTT and 5% of maximal packet loss, key deployment for a grid topology of 50 nodes would require with TOPKEY about 16 seconds and for 100 nodes it would take up to 1 minute. Considering measurements for small topologies from Figure 2 and heuristic analysis for larger networks from Figure 3 we conclude that key deployment with TOPKEY can be performed fully automated within a reasonable amount of time for networks of medium size.

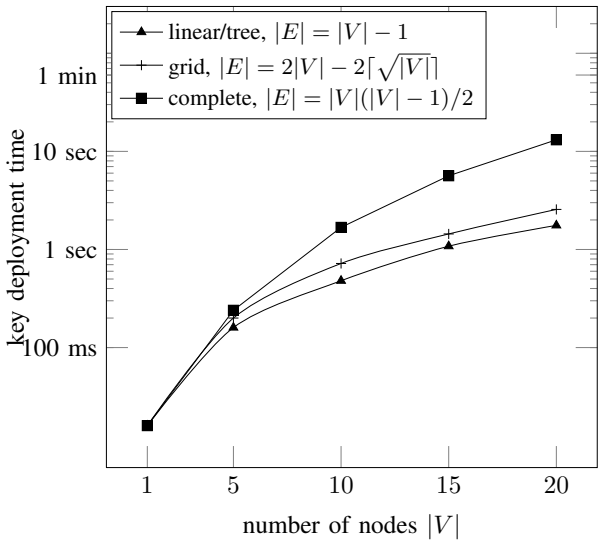

Fig. 2. TOPKEY Key Deployment Performance

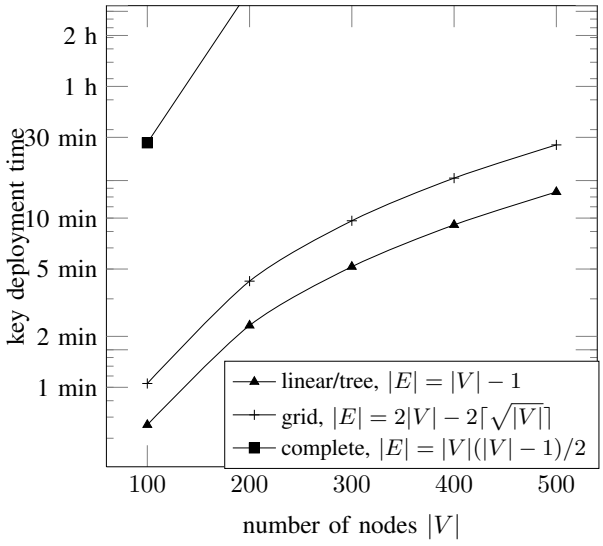

Fig. 3. Performance Heuristics for Larger Networks

VIII. Conclusion

Our TOPKEY framework offers tool assistance for secure initialization of commodity sensor nodes and comes with a fully automated and fast key deployment procedure, where reduction of transmission power is used as a countermeasure against eavesdropping attacks. Our experimental analysis on TelosB nodes with Contiki OS suggests that this approach may offer sufficient level of protection when eavesdropper's receiving abilities do not exceed those of sensor nodes and where the attacker can be kept out of the estimated secure ranges, e.g. in commodity office or home environments. The modularity of TOPKEY may further support dynamic WSN topologies, in which case the topology design step can be skipped and the tool would have to be extended with implementation of topology-unaware key distribution algorithms, e.g. [44]. Our TOPKEY tool can be downloaded as open-source project from [1].

References

[1] TOPKEY on GitHub, https://www.github.com/aiQon/topkey, May 2012.

[2] G. Ács, L. Buttyán, and I. Vajda. Modelling Adversaries and Security Objectives for Routing Protocols in Wireless Sensor Networks. In *4th ACM SASN*, pages 49–58. ACM, 2006.

[3] G. Alder. *Design and Implementation of the JGraph Swing Component*, 1.0.6 edition, Feb. 2003. Available at: http://jgraph.sourceforge.net/doc/paper/.

[4] R. Anderson, H. Chan, and A. Perrig. Key infection: Smart trust for smart dust. In *IEEE ICNP 2004*, pages 206–215. IEEE, 2004.

[5] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In E. Brickell, editor, *CRYPTO*, volume 740 of *LNCS*, pages 471–486. Springer Berlin / Heidelberg, 1993.

[6] C. Castelluccia, A. Chan, E. Mykletun, and G. Tsudik. Efficient and Provably Secure Aggregation of Encrypted Data in Wireless Sensor Networks. *ACM TOSN*, 5(3):20, May 2009.

[7] C. Castelluccia and P. Mutaf. Shake them up!: A movement-based pairing protocol for cpu-constrained devices. In *MobiSys*, pages 51–64, New York, NY, USA, 2005. ACM.

[8] A. C.-F. Chan and C. Castelluccia. A Security Framework for Privacy-preserving Data Aggregation in Wireless Sensor Networks. *ACM Transactions on Sensor Networks (TOSN)*, 7:29:1–29:45, February 2011.

[9] H. Chan, A. Perrig, and D. Song. Random Key Predistribution Schemes for Sensor Networks. In *IEEE S& P*, pages 197–213. IEEE CS, 2003.

[10] H. Chan, A. Perrig, and D. Song. Secure Hierarchical In-Network Aggregation in Sensor Networks. In *ACM CCS*, pages 278–287. ACM, 2006.

[11] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks. *ACM TISSEC*, 8:228–258, May 2005.

[12] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *IEEE LCN*, pages 455–462. IEEE CS, 2004.

[13] L. Eschenauer and V. D. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In *ACM CCS*, pages 41–47. ACM, 2002.

[14] J. Girao, D. Westhoff, E. Mykletun, and T. Araki. TinyPEDS: Tiny Persistent Encrypted Data Storage in Asynchronous Wireless Sensor Networks. *Ad Hoc Networks*, 5(7):1073–1089, Sept. 2007.

[15] L. Hu and D. Evans. Secure Aggregation for Wireless Networks. In *SAINT-W*, pages 384–391. IEEE CS, 2003.

[16] J. Hwang and Y. Kim. Revisiting Random Key Pre-distribution Schemes for Wireless Sensor Networks. In *ACM SASN*, pages 43–52. ACM, 2004.

[17] C. Karlof and D. Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. In *IEEE International Workshop on Sensor Network Protocols and Applications 2003*, pages 113–127, 2003.

[18] C. Kuo, M. Luk, R. Negi, and A. Perrig. Message-in-a-bottle: user-friendly and secure key deployment for sensor nodes. In *SenSys*, pages 233–246. ACM, 2007.

[19] B. Lai, S. Kim, and I. Verbauwhede. Scalable session key construction protocol for wireless sensor networks. In *IEEE LARTES*. IEEE, 2002.

[20] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure Dissemination of Code Updates in Sensor Networks. In *IEEE ICDCS*, page 53, 2006.

[21] J. Lee and D. Stinson. Deterministic key predistribution schemes for distributed sensor networks. In *Selected Areas in Cryptography*, pages 294–307. Springer, 2005.

[22] P. A. Levis. Tinyos: An open operating system for wireless sensor networks (invited seminar). In *7th International Conference on Mobile Data Management*, page 63. IEEE CS, 2006.

[23] M. Li, S. Yu, W. Lou, and K. Ren. Group device pairing based secure sensor association and key management for body area networks. In *INFOCOM*, pages 2651–2659. IEEE Press, 2010.

[24] D. Liu and P. Ning. Location-based pairwise key establishments for static sensor networks. In *1st ACM workshop on Security of ad hoc and sensor networks*, pages 72–82. ACM, 2003.

[25] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. *ACM Transactions on Information and System*, 2005.

[26] M. Manulis and J. Schwenk. Security model and framework for information aggregation in sensor networks. *ACM TOSN*, 5(2), 2009.

[27] R. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[28] T. Perkovic, M. Cagalj, T. Mastelic, N. Saxena, and D. Begusic. Secure initialization of multiple constrained wireless devices for an unaided user. *IEEE TMC*, PP(99):1, 2011.

[29] T. Perkovic, I. Stancic, L. Malisa, and M. Cagalj. Multichannel Protocols for User-Friendly and Scalable Initialization of Sensor Networks. In *SecureComm*, pages 228–247, 2009.

[30] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. Spins: Security protocols for sensor networks. In *MobiCom*, pages 189–199, 2001.

[31] S. Peter, D. Westhoff, and C. Castelluccia. A survey on the encryption of convergecast traffic with in-network processing. *IEEE TDSC*, 7(1):20–34, 2010.

[32] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN*, pages 364–369. IEEE Press, April 2005.

[33] B. Przydatek, D. X. Song, and A. Perrig. SIA: Secure Information Aggregation in Sensor Networks. In *SenSys*, pages 255–265. ACM, 2003.

[34] L. B. Ruiz, J. M. Nogueira, and A. A. F. Loureiro. MANNA: a management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–125, 2003.

[35] N. Saxena and M. B. Uddin. Automated device pairing for asymmetric pairing scenarios. In *ICICS*, volume 5308 of *LNCS*, pages 311–327. Springer, 2008.

[36] N. Saxena and M. B. Uddin. Blink 'em all: Scalable, user-friendly and secure initialization of wireless sensor nodes. In *CANS*, volume 5888 of *LNCS*, pages 154–173. Springer, 2009.

[37] N. Saxena, M. B. Uddin, and J. Voris. Universal device pairing using an auxiliary device. In *SOUPS*, pages 56–67. ACM, 2008.

[38] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *ACM WiSe*, pages 85–94. ACM, 2006.

[39] O. Ugus, D. Westhoff, and J.-M. Bohli. A rom-friendly secure code update mechanism for wsns using a stateful-verifier tau-time signature scheme. In *ACM WiSec*, pages 29–40. ACM, 2009.

[40] D. Westhoff, J. Girão, and M. Acharya. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation. *IEEE TMC*, 5(10):1417–1431, 2006.

[41] A. D. Wood, L. Fang, J. A. Stankovic, and T. He. SIGF: a family of configurable, secure routing protocols for wireless sensor networks. In *ACM SASN*, pages 35–48. ACM, 2006.

[42] Z. Yu and Y. Guan. A Key Management Scheme Using Deployment Knowledge for Wireless Sensor Networks. *IEEE TPDS*, 19:1411–1425, October 2008.

[43] S. Zhu. LEAP : Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks Categories and Subject Descriptors. In *ACM CCS*, pages 62–72, 2003.

[44] S. Zhu, Sencun and Setia, Sanjeev and Jajodia. LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. *ACM TOSN*, 2(4):500–528, 2006.