Chapter 1

TinyTO: Two-way Authentication for Constrained Devices in the Internet-of-Things

Corinna Schmitt, Martin Noack, Burkhard Stiller

1.1 Abstract

Wireless Sensor Networks (WSN) will play a fundamental role in the future Internet-of-Things (IoT), with millions of devices actively exchanging confidential information with each other in a multi-hop manner. Ensuring secure end-to-end communication channels is crucial to the success of innovative IoT applications, since they are essential to limit attacks' impacts and avoid exposure of information. End-to-end security solutions, like IPsec or DTLS, do not scale well on WSN devices due to limited resources. In this chapter the optimized two-way authentication solution for tiny devices (TinyTO) combines end-to-end secured communication with WSN design. TinyTO provides confidentiality and integrity within a fast and secure handshake, works with public-key cryptography, and uses Elliptic Curve Cryptography (ECC) for message encryption and authentication. ECC lowers the resource consumption and suits devices with 10 kByte RAM and 100 kByte ROM. TinyTO does not need a network-wide shared secret, it is application-independent, and it supports in-network aggregation. [1]

[1] This book chapter's content is based on the Master Thesis [55] performed at the Communication Systems Group of the University of Zurich, Switzerland.

Keywords: Wireless Sensor Network (WSN), Internet-of-Things (IoT), constrained devices, end-to-end-security, two-way authentication, Elliptic Curve Cryptography (ECC)

1.2 Introduction

Atzori et al. already stated in 2010 that the Internet-of-Things (IoT) consists of manifold devices ranging from IP networks and servers to small devices like Wireless Sensor Network devices (*e.g.*, Radio-Frequency IDentification (RFID) tags or sensor nodes) [1]. Throughout the years especially Wireless Sensor Networks (WSN) consisting of constrained devices with limited resources in memory, energy, and computational capacity rapidly gain popularity. Thus, the questions raised how to integrate them into the IoT and what challenges occur looking on their constrained resources [2], [3], and [4]. The number of possible deployments of such networks rises and more applications have a need for confidential and authenticated communication within the network. This security issue must be addressed due to the fact that sensitive information (*e.g.*, Identity (ID), names, or Global Positioning System (GPS) information) is linked almost everywhere to all kinds of collected data, like temperature, sound, and brightness [5], [6], and [7]. Hence, collected data is no longer anonymous and often desired to be kept confidential. Figure 1 illustrates this case for a building scenario, where environmental data is collected in rooms and transmitted over multiple hops to the gateway in order to make the data available to applications, such as the climate control, security office, and room calendar (cf. Section 1.6). If room information can be retrieved by eavesdropping due to missing security in the communication, an attacker would be aware of sensitive

information and could plan for example a burglary. Therefore, collected data must be transmitted in a secure manner and/or over a secure channel providing end-to-end security, giving only authorized entities (*e.g.*, gateway, security system, or company members) access to this confidential information. But how is this supposed to be done? Keeping in mind that WSNs are part of the IoT and consist of constrained devices with limited resources, any security risks are aggravated by WSN design and security requirements of the IoT. Ultimately, an end-to-end security solution is required to achieve an adequate level of security. Protecting data only after it leaves the scope of the local network (*e.g.*, WSN) is not sufficient.

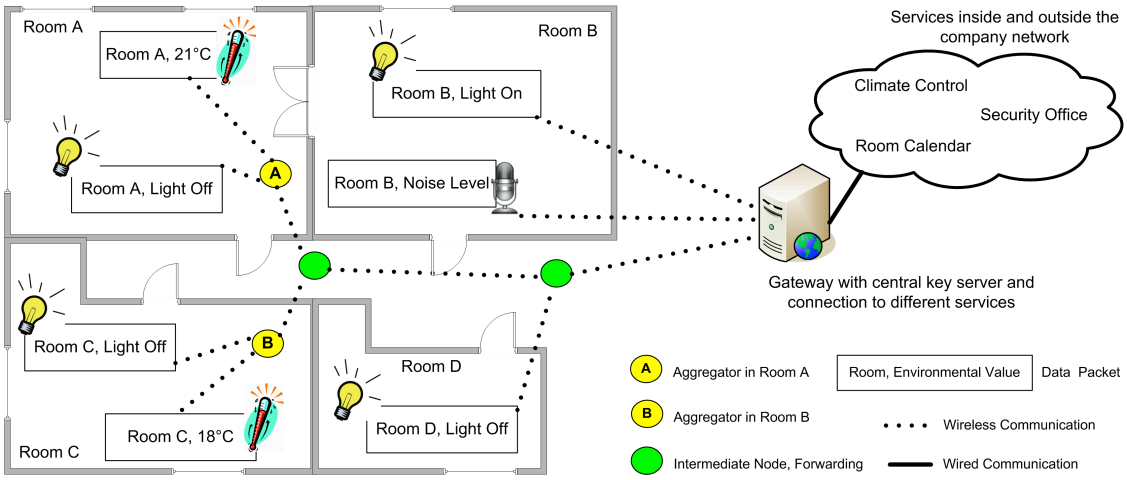

Figure 1: Building Scenario.

Using existing technologies (*e.g.*, Secure Sockets Layer (SSL)/Transport Layer Security (TLS) [56] or cryptography [57, 58]) is the easiest way to achieve the goal of secure data transmission. But this becomes increasingly challenging when looking at WSN devices used today (*e.g.,* RFIDs, heart beat monitor, or environmental sensors), since their resources are strictly limited in memory, power, and computational capacity [8] and [9]. Those WSN devices are divided into constrained classes corresponding to

their computational capacity and memory resources (cf. Table 1) [8]. Security support is very challenging when assuming class 1 devices (*e.g.*, TelosB [10]), as done for the proposed solution TinyTO, because they offer only about 10 kByte RAM and 100 kByte ROM. A standard approach for securing communications in the Internet is SSL/TLS [56] relying on asymmetric cryptography like RSA requires many resources and computational capacity and, thus, it is only feasible for at least class 2 devices (approximately with 50 kByte RAM and 250 kByte ROM) [11]. An additional challenge is the device diversity in today's WSNs, the network size itself, and multiple requirements (*e.g.,* lifetime or security support) due to the target application [3]. Developing a proper solution is still a challenge, especially for security issues under consideration of the aforementioned challenges and constraints. Depending on the application, it might be prohibited to reuse existing solutions (*e.g.*, military area). In general it is preferred to reuse standards or to develop a generic solution that can be integrated without major modifications and shall not require hardware features, like cryptographic coprocessors [11], certain radio modules, or specific processors. On the software side, it shall not require a specific protocol stack, but it shall rely on the most basic interfaces and be kept separated from applications in order to allow simple integration into any used protocol stack with a limited number of connection points (*i.e.*, interfaces). Furthermore, all additional features have to avoid affecting excessive performance and memory consumption.

Table 1: Device Classes (1 KiB = 1024 Byte). [8]

Name	RAM	ROM	IP Stack	Security
Class 0	<< 10 KiB	<< 100 KiB	NO	NO
Class 1	~10 KiB	~100 KiB	CoAP [22], BLIP [59, 60]	YES
Class 2	~50 KiB	~250 KiB	HTTP, TLS	YES

Based on the aforementioned hardware and application requirements the proposed security solution TinyTO, an optimized two-way authentication solution for tiny devices, provides confidential data transfer with an additional integrity protection and data authentication as well as a two-way authentication between sender and receiver of messages, delivering end-to-end security even for class 1 devices. This is achieved by introducing an efficient handshake with a direct authentication and key-exchange between pairs of nodes in the network, setting up an encrypted data transfer with an integrated encryption scheme. To minimize overall hardware requirements, the Elliptic Curve Cryptography (ECC) is used for key generation, key exchange, encryption, decryption, and signature generation.

Initially, each node is only familiar with the gateway. This relationship is authenticated with an individual shared key (in TinyTO of 16 Byte length), which is only known to the gateway and the node, and is deployed to all nodes during the initial programming routine. Individual keys between nodes are established during the handshake performance or can be requested by a node from the gateway (*e.g.*, in case of communication with aggregator).

TinyTO is designed to fit WSN requirements, is application independent, and allows for an easy integration into existing applications due to its modular nature. TinyTO explicitly supports in-network aggregation by enabling a full and secure end-to-end communication without the need for a network-wide shared secret.

In the following, all data that is transmitted in data packets is considered to be confidential. The remainder of this chapter is structured as follows. Section 1.3

introduces relevant work in the area of Pre-shared Keys (PSK), ECC usage, and authentication without any special requirements to infrastructure. Afterwards, Section 1.4 presents the design decisions for TinyTO followed by a detailed description of the proposed solution TinyTO within Section 1.5. The approach is evaluated in Section 1.6 in respect to resource consumption, run time performance, and security aspects. Finally, Section 1.7 summarizes the chapter.

1.3 Security Aspects and Solutions

The necessity to provide an end-to-end security solution in WSNs is not entirely new. Over the years different approaches have emerged addressing various security issues.

Thus, an often quoted solution is pre-distributing symmetric keys. However, flexibility of the deployment, connectivity between nodes, and resilience against attackers is limited significantly [12]. Instead, Du et al. proposed a solution that applies public key authentication with smaller resource demanding symmetric key operations, where a one-way hash function is used to authenticate public keys. The basic idea is to allow for individual nodes to verify that a transmitted public key matches the claimed identity, without relying on a trusted third party (*e.g.*, Certificate Authority (CA)). For an exhausting mapping between all keys and identities a large number of keys and certificates must be stored on every node, which is not feasible. Hence, a hash function mapping from identity to the hash value of the corresponding public key is pre-shared. Thus, only hash values and identities must be compared, which requires only a fraction of the memory and computational power. This can be optimized further by using Merkle Trees, where non-leaf nodes are labeled with the hash of those labels of its children [13].

ECC determines a promising option for WSN security solutions, in particular for message encryption, since ECC can deliver strong security with only a small amount of resources needed, as denoted in [14], [15], and [16]. A 192 bit ECC key provides the same level of security as a RSA-key in range of 1024 bit to 2048 bit [17]. ECC is viable for key generation, key exchange, encryption, decryption, and signatures, especially in resource constrained applications.

Nie et al. developed the HIP DEX protocol for hop-by-hop secure connections using a Diffie-Hellman key exchange for public keys and the AES encryption for the session key exchange [18]. Computational requirements are reduced by limiting cryptographic primitives to a minimum (*e.g.*, removing expensive signature algorithms and any form of cryptographic hash functions). Cryptographic challenges are included in the first messages of the handshake proposed, in order to avoid flooding attacks. Identity authentication is achieved by password verification within the handshake, where nodes need to know their respective passwords in advance.

The PAuthKey protocol for application-level end-to-end security overcomes the problem of two-way authentication (i.e. mutual authentication) between sensor nodes [19]. It provides pervasive lightweight authentication and keying mechanisms, allowing nodes to establish secure and authenticated communication channels with each other. PAuthKey employs ECC-based implicit certificates, using a trusted central CA to handle authentication security. Thus, it stands in contrast to other authentication approaches, since certificates are generally considered to be resource challenging for WSNs and they require additional hosting infrastructure (*e.g.*, CA) or hardware (*e.g.*, Trusted Platform Module (TPM)) that can be integrated on the gateway or as external network entity.

The UbiSec&Sens project offered a tool-box of security-aware components. The proposed Zero Common Knowledge (ZCK) protocol for authentication can establish well-defined pairwise security associations between entities, even in the absence of a common security infrastructure and pre-shared secrets [20]. ZCK authentication is based on re-recognition between entities, allowing entities to authenticate any other entity known from the past. This approach does not provide full security, as required, for instance for financial transactions, since the first contact between entities cannot be authenticated. However, in a scenario without any form of pre-established knowledge or a trusted third party, ZCK provides the best level of security that can be achieved under those limitations given. The ZCK protocol itself does not cater for a key exchange, but can be used in combination with any form of cryptography, like Diffie-Hellman [21].

TinyDTLS – a DTLS-based solution for constrained (tiny) devices – provides end-to-end security, but targets class 2 devices with additional memory resources [11]. In this case the platform used includes a TPM, offering additional dedicated memory and computational power for costly security functions. TinyDTLS performs a TLS handshake, using X.509 certificates for authentication and Advanced Encryption Standard (AES) for encryption, but still exceeds most alternatives due to the high amount of available resources on its target devices. An advantage of this solution is the compatibility with established standard protocols such as SSL/TLS [56].

The security aspects addressed by TinyTO are a direct result of the aforementioned existing solutions and the final design decisions taken in the upcoming Section 1.4, especially to counter Unknown Key-Share Attacks (UKSA) and Man-In-The-Middle (MITM) attacks. Therefore, TinyTO's goals are summarized as:

1. TinyTO brings end-to-end security to class 1 devices by providing two-way authentication.
2. The TinyTO's handshake design with two-way authentication adds immensely to the security level without an involvement of certificates and certificate authority (CA) in the network's infrastructure or special hardware components like TPM on the device.
3. TinyTO is protected against MITM attacks in contrast to other solutions for class 1 devices like UbiSec&Sens and ZCK.
4. TinyTO allows for adding devices dynamically to the secure network in contrast to static Merkle Trees.
5. TinyTO uses the Routing Protocol for Low power and Lossy Networks (RPL) [60, 41], offering various measurements to improve routing, which can be used for an attack detection and defense.

In order to address these goals TinyTO requires pre-programmed master keys for authentication between devices and the gateway, RPL routing, and a support of an ECC functionality for encryption and signing.

1.4 Design Decisions

An ideal solution for the two-way authentication should work generically on WSN nodes of all classes, especially since the trend goes towards heterogeneous WSNs. However, since WSN nodes are primarily designed to collect data, they prioritize cheapness and long lifetime over processing power and memory size. Section 1.2 outlined that class 1 devices are per definition in RFC 7228 [8] very constrained to run security schemes

beyond very specific implementations mentioned in Section 1.3. Thus, the newly proposed end-to-end security solution in this book chapter targets class 1 devices as a minimum requirement. Even though class 1 devices can connect to the Internet without additional proxies or gateways, they are limited in communications with peers, if those peers have a full protocol stack employed [8], which would overwhelm available resources of class 1 devices. Therefore, class 1 devices require a specifically designed protocol stack for constrained devices, like the Constrained Application Protocol (CoAP) over User Datagram Protocol (UDP) [22]. Consequently, traditional security concepts for wireless networks, such as Wired Equivalent Privacy (WEP) or TLS in their native form, are unsuitable for WSNs as pointed out in [23].

One approach to adapt the traditional Public Key Cryptography (PKC) to WSNs (cf. Section 1.3) is the integration of extra hardware into nodes [12] for performing security operations and operating separated from main application and the node processor. At a first glance, class 2 devices have more resources and can be used for this purpose [8]. Among other functionality class 2 devices can deliver Internet-level security by providing confidentiality and message authentication at high speed [11]. Hu et al. have shown that a TPM chip outperforms most alternative solutions of similar resource levels [24]. But on the second glance, as a drawback all nodes in a WSN need to be equipped with an appropriate amount of resources (*e.g.*, more RAM/ROM or using a TPM) to apply the security scheme network wide.

A class 1 device cannot build and maintain a RFC-compliant Public Key Infrastructure (PKI), while executing its main task – data collection and data forwarding – that is already resource consuming in itself. One commonly used OpenSSL X.509 RSA-1024 certificate alone has a size of about 800 Byte [25] plus the corresponding RSA

key pair takes additional 800 Byte [26]. Assuming an aggregation support, *n+2* certificates and *n+2* key pairs for a degree of aggregation (*doa*) of n must be stored, quickly filling the available memory. For example, following those calculations, an aggregator with *doa = 5* needs to store additional 11.2 kByte of data, only for certificates and corresponding key pairs.

This extreme memory consumption can be avoided by utilizing PKC only between designated node pairs (cf. Section 1.3), so that every node (aggregator or collector) only has to store its own key pair and the public key of the given recipient (i.e., gateway or next hop). Gura et al. showed the general feasibility of PKC on simple 8-bit processors as typically found within WSN nodes [27]. Therefore, TinyTO's security solution is based on PKC. Furthermore, memory and energy consumption savings are gained by applying ECC instead of RSA (Rivest, Shamir, and Adleman) for key generation, key exchange, signatures, and encryption. The National Institute of Standards and Technology (NIST) recommendations SP 800-57 explain that a RSA key in range of 1024 bit to 2048 bit delivers the same security level as a 160 bit ECC key, *i.e.* the same amount of resources is required to break them [17]. Even more, Arvinderpal et al. showed that ECC implementations are faster and require less energy compared to equally secure RSA algorithms [26].

In general, standardization bodies and researchers agree on a set of security objectives that are necessary to achieve information security: Confidentiality, integrity, authenticity, availability, and accountability of all messages as defined in [28], [29], and [30]. Furthermore, a set of requirements that are particular to WSNs and the development goals for TinyTO must be considered: (1) End-to-end security to avoid eavesdropping and spoofing attacks meaning risk for the communication, because the underlying

network infrastructure is only partially under the user's control and might be compromised. Especially in a WSN, where multi-hop communications are common, authentication and key exchange is an essential design goal. (2) In WSNs connections are often not lossless. TCP (Transmission Control Protocol) erroneously invokes congestion control mechanisms to counter the loss of packets, which drastically impacts the performance and results in the UDP to serve as a better choice for WSNs [31]. (3) Two-way authentication denotes two entities authenticating each other at the same time [32]. In the scope of WSNs, it is not sufficient to authenticate only the sender to the receiver, but the sender has to be sure also about the identity and authorization of the potential receiver of confidential information. (4) ECC is promising to save resources, when performing PKC in TinyTO. For message encryption an Integrated Encryption Scheme (IES) is applied, especially to harness the speed-advantage of symmetric encryption for large amounts of data without the drawback of a repeated key exchange for every transmission, which otherwise is necessary so that no secret credential is used more than once.

Diffie et al. argued that an authentication protocol should always be linked to the key exchange for later encryption; otherwise an attacker might just wait until the authentication is completed to compromise the established communication channel thereafter [33]. Canetti et al. summarized the objective of a key exchange protocol in a very intuitive way: A key exchange protocol is secure, if it is impossible or at least infeasible for an attacker to distinguish the generated key from a random value [34]. The same fundamental concept can be applied to the Authenticated Key Exchange (AKE) protocol. But additionally, entity (or party) authentication has to guarantee the identity of communicating parties in the current communication session and, therefore, has to

prevent impersonation [35]. A good authentication protocol combines several properties as explained by various researchers [36], [37], [38], [39], and [33] and is relevant for TinyTO's design: (1) Forward secrecy guarantees such that, if a generated private key of one or more of the participating entities is compromised, the security of previous communications is not affected. (2) Asymmetry of messages is required to avoid reflection attacks, where one entity simply replays the same message back to the sender; it is desirable to avoid symmetries. In other words, the authentication responses of two different parties must not be identical. (3) Direct authentication is provided by a protocol, if the authentication is complete in a successful handshake, *i.e.* both parties have proved knowledge of the shared secret. (4) Timestamps are to be avoided, because not every participating entity can be expected to maintain a reliable local clock, which must be synchronized periodically, too.

1.5 The TinyTO Protocol

Due to TinyTO's main goal to support an end-to-end security with two-way authentication on class 1 devices, the authentication protocol has to include always a key exchange, such that several possible handshake candidates can be considered in practice, leading to the final design and implementation of TinyTO. First, handshake candidates for TinyTO and their drawbacks are introduced. Second, the resulting TinyTO handshake including two-way authentication purposes and aggregation support are described. Finally, key information on the respective implementation is presented.

Possible Handshake Protocol Candidates

Handshake protocol candidates considered in this section support a two-way authentication of two independent entities without prior information exchange, which make them highly appropriate for TinyTO. From this stage on, the traditional naming pattern of cryptography is applied for protocol descriptions assuming two communication parties – Alice and Bob –, which are instantiated as sensor nodes.

At a first glance the Station-to-Station protocol (STS) seems to be an ideal candidate for TinyTO, because STS is based on a Diffie-Hellman's key exchange, followed by an exchange of authentication signatures [35]. Both parties Alice (A) and Bob (B) compute their private key x and a public key X in the beginning. Next, Alice sends her public key X_A to Bob. Once Bob receives X_A, he can compute a shared secret K_{AB} with X_A and x_B, according to the Diffie-Hellman's key exchange algorithm [32]. Bob can now encrypt any message to Alice using K_{AB}. For decryption purposes Bob sends X_B back to Alice, so that she can compute the same shared secret K_{AB}. Additionally, Bob sends a token consisting of both public keys, signed with his own private key to authenticate himself. Alice can use X_B to verify that Bob was indeed the same person, who signed the message and computed the shared secret. Bob is now authenticated to Alice. As the last step of the two-way authentication Alice constructs an authentication message and sends it to Bob to authenticate herself to Bob. To avoid unnecessary communication overhead, the second key exchange message is combined with the first authentication message. As a result, STS entails the establishment of a shared secret key between two parties with mutual entity authentication and mutual implicit key authentication [32]. The forward secrecy can be provided by deriving a new ephemeral

key from the shared secret for the encryption of every message in that exchange [40]. The signatures are used to obtain protection against impersonation during the exchange.

However, there are two main shortcomings: (1) While the STS is relatively simple to execute, it does not include any explicit key confirmation. Neither Bob nor Alice inherently can be sure that the other party has actually computed a shared secret without additional messages. (2) Furthermore, STS is vulnerable to UKSAs and the MITM attack [35]. To prevent UKSAs and to provide explicit key authentication, the signatures used can be encrypted additionally with the successfully computed K_{AB} [33]. Thus, Bob is assured that he shares K_{AB} only with one single party, namely Alice. Since he has created X_B specifically for this handshake and Alice has signed X_B and X_A, her signature is now tied to this particular handshake. By encrypting the message with the resulting K_{AB}, Alice assures Bob that she was indeed the entity, who created X_A. Similar assumptions can be made from the position of Alice [33]. This modification requires more computational capacity, due to parallel execution of signature and symmetric encryption algorithms. Hence, for WSN devices below class 2, it is desirable to avoid this sort of overhead. The need for encryption can be resolved by including the identity of both parties in the exchanged signatures resulting in the adapted STS protocol [40]. When combining the adapted STS with identities in signatures it becomes almost functionally identical to the Bellare-Canetti-Krawczyk protocol (BCK) [37], [36], [40]. The only difference in BCK is the absence of the sending parties' identities. According to Basin et al., it is generally desirable to include identities of both parties to avoid the spoofing of identities [42]. But in a bidirectional exchange, as it is the case for BCK, it is only required to include the receiver's identity [42]: At least in one direction, the receiving party is presented with an

invalid signature that does not contain its own identity and as a result immediately abort the handshake.

At this point, BCK is computationally relatively cheap, but still vulnerable to MITM attacks [40]. This weakness boils down to the fact that it is impossible to reliably map a public key to a specific entity, *i.e.* to derive their public key from their identity. Any party can claim any public key as their own. To counteract, it is essential to strongly couple a public key with the respective identity. The prevalent solution for this is to introduce a PKI with certificates and trusted CAs as proposed for TLS [43]. A certificate contains the identity and the corresponding public key. Entities can be assured of the correct coupling between key and identity, because trusted CAs constructed the certificate. However, BCK itself does not suit the given requirement of class 1 devices, but can be used as a baseline as justified in the upcoming section.

BCK with Pre-shared Keys for TinyTO

Badra et al. have outlined that PSK is suitable to provide authentication [44], while requiring only a small amount of computational power and memory. Thus, it is selected for TinyTO to verify the identity of an entity. The distribution of PSKs is simple in the context of WSN devices: Adding a unique PSK to the programming procedure introduces practically no overhead, since nodes need to be programmed before deployment in any case and the key generation and management can be moved to the software programming the nodes. Compared to approaches where every node is equipped with a set of keys for encryption between peers before deployment, TinyTO assumes that every node has only one PSK, solely for authentication toward the gateway. The developed handshake for TinyTO compares to BCK with pre-shared keys that form master keys for an initial

authentication between the node and the gateway. Figure 2 illustrates the resulting handshake, where Alice and Bob can represent anyone of the following device types in the WSN:

- A collector is a device collecting sensor, which forwards them directly to the next device in communication range.
- An aggregator works with the data received either as aggregating several messages into one large message or pre-processes data (*e.g.*, average, max, min calculation of values) before forwarding them to the next device in communication range.
- The gateway defines the gate to the world connecting the WSN with other applications in the IoT domain.

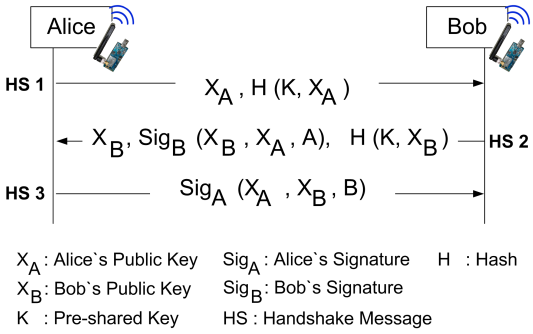

Figure 2: Extended BCK protocol with PSK for TinyTO.

Under the assumption that only the two parties under investigation have knowledge of the PSK, each party can be assured that indeed the other communication party uses this PSK. It is vital not to transmit the PSK in plaintext during the authentication in order to keep the PSK a secret between the two parties. Otherwise any attacker, who picks up that message containing the PSK, can use the PSK. Thus, it must be avoided to send any form of information that can (a) be used to retrieve the PSK or (b)

replayed to achieve authentication for any other entity. Traditionally, those two goals are met by transmitting a cryptographic hash digest of the PSK together with a cryptographic nonce [45]. Including a different nonce in every message makes it impossible to reuse an authentication message (*e.g.*, replay-attack). In comparison, TinyTO desires to couple a unique public key with the PSK (and, thus, the identity), which may be replayed several times, but never for another public key that makes it very hard to recalculate the PSK by an attacker. Hence, it is possible to use the public key instead of a random nonce and to create a hash from the PSK and this public key, *i.e.* $H(K,X_A)$. This ensures Bob that X_A is indeed Alice's public key [46] and [47]. A cryptographic hash function is infeasible to be reverted, even with a partially known input (the public key is obviously publicly known). But the PSK is not recoverable [47]. At the same time, a spoofed hash digest for a different public key can be produced only with the knowledge of the PSK. To provide mutual authentication in the TinyTO protocol, those digests must be computed from both parties, with their respective public keys. For avoiding transmission overhead, these digests can be included in the first and second handshake messages (HS1 and HS2 in Figure 2) in order to avoid any transmission overhead by additional messages.

In accordance to the requirements for TinyTO, this approach determines the two-way authentication protocol, which includes as key agreement delivering a direct and explicit key authentication [55]. Messages do not include timestamps, they are completely asymmetrical, and they cannot be used for a replay or reflection attacks. Appropriate encryption techniques (*e.g.*, RSA or AES) of subsequent messages are required to guarantee the forward secrecy beyond the handshake.

As explained in Section 1.4, two flexible roles – collector and aggregator – are possible for a node. The gateway, in contrast, is unique and static. Collectors and

aggregators use TinyTO to establish a secure communication channel with the gateway. Aggregators introduce additional performance overhead to TinyTO and the WSN, because the handshake is more complicated (see Figure 3). Also, the collectors need to switch the destination of their data stream from the gateway to the aggregator, which in turn needs to process the information. Therefore, the aggregator sends a presence announcement via a broadcast to collectors that redirect their streams upon receipt. Schmitt et al. stated that four conceptual steps are required for an aggregator introduction, if no authentication is required [9]. The TinyDTLS solution [11] inspired the development of TinyTO. [11] specifies four steps that must be taken to establish a two-way authentication and those must be slightly adapted for the proposed TinyTO solution in the following manner: (1) Collectors complete their TinyTO handshake with the gateway (cf. Figure 2) and transmit data over a secure channel. (2) In turn, the aggregator can be activated, contacting the gateway immediately and executing the TinyTO handshake resulting in a secure channel. (3) The aggregator broadcasts its presence to collectors in range that are programmed to wait for such a specific message type (*e.g.*, simple echo request, counter, or nonce). The aggregator's public key is included in the broadcast message to avoid additional message exchanges. (4) Finally, collectors redirect their streams to the aggregator, encrypted with the aggregator's public key ($E\{M\}_{XA}$). The aggregator decrypts incoming streams, processes messages, encrypts the results again, and sends the new message securely to the gateway ($E\{M\}_{XG}$) or the next hop.

While the above described approach of aggregator integration provides an encryption of messages between all parties and, therefore, a protection against eavesdropping between collectors and the aggregator as well as the aggregator and the gateway, it entails one important drawback: Collectors have executed the complete

TinyTO handshake with the gateway, resulting in a two-way authentication of both parties. However, in step (3) and (4) above collectors sacrifice all assertions about identities, if they blindly react to aggregator's broadcasts. Attackers just need to broadcast an aggregator announcement to reach access to data streams from every collector in range that are conveniently encrypted with the attacker's own public key. Since such a situation breaks the entire security concept, collectors are requested to establish a new secure channel to aggregators that fulfill TinyTO's design principles without blindly trusting aggregator broadcasts. Consequently, the authentication needs to be extended by an authorization: Collectors need the confirmation that a broadcasting aggregator is a valid aggregator and not an intruder, who tries to access confidential information.

Assuming that collectors or aggregators communicate only with the gateway, the request for secure communication is implicitly covered by the exchange of pre-programmed PSKs. Intuitively, it is possible to pre-program aggregators and collectors with pairwise PSKs in the same way, followed by a handshake execution, including the authentication and the key exchange. But this workaround does not fulfill the flexibility requirement for TinyTO: In this case, aggregators can only aggregate data streams from pre-defined collectors and will further need to hold $n+1$ PSKs for n collectors. A more flexible and less resource demanding solution is to lever the already fully authenticated and secured channels between both aggregator and collector, and the gateway (cf. Figure 3). Upon the receipt of an aggregator announcement, collectors need to check only with the fully authenticated gateway, if the broadcast sender is an authorized aggregator. If so, the gateway can reply with the aggregator's public key signed by his own trusted private key. This is similar to a PKI, where the gateway takes the role of the

certificate authority as a trusted third party. For TinyTO it is assumed that all parties have completed previously and successfully their handshakes with the gateway. The signature is used in order to verify the mapping between the aggregator's public key and its identity, which makes spoofing attacks on the public key impossible. This stands in contrast to the previously exchanged authentication messages, where the identity of the receiving party must be included instead of the key owner's identity, because the channel between gateway and collector is already authenticated. The aggregator's identity is not encrypted between the collector and the gateway allowing for spoofing attacks on the identity, since expensive computations would be required for this additional encryption. Hence, it is substituted with the identity in the signature from the next message, computed by the more powerful gateway. The gateway might reply with the public key for a spoofed identity, but it is detectable by the collector due to the invalid signature, resulting in the process' abortion.

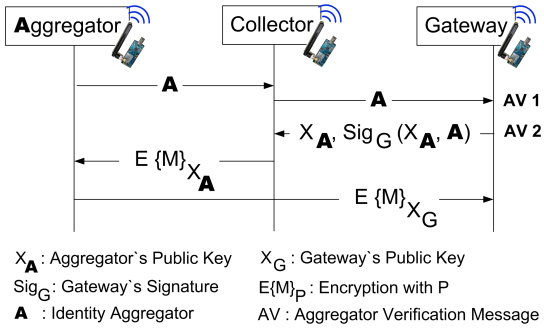

Figure 3: Secure aggregation support.

Handshake Implementation

From now on it is assumed that TinyTO is implemented in TinyOS, where different components are "wired" to each other and using the offered set of functionality. Thus, the TinyTO handshake is implemented in the component HandshakeHandler, which is

exclusively responsible for handshake messages handling, including message composition and reply handling. HandshakeHandler is wired to components called for cryptographic functions. The three TinyTO handshake messages HS1 to HS3 (cf. Figure 2) are implemented in a similar manner. Listing 1 shows exemplary the structure of handshake message HS2 where *nx_ uint8_ t* stands for the network-serializable unsigned integer type.

The *msgType* field is used to distinguish between handshake messages and other types of control messages that are sent via the same port. *hsType* identifies different handshake messages HS1, HS2, and HS3. Furthermore, public ECC keys are decomposed into x and y-coordinates [33] and [35] for easy handling on the node`s side. Elliptic Curve Digital Signature Algorithm (ECDSA) signatures are defined as integer key pairs, written as *(r, s)*, and, therefore, difficult to include in a fixed-length packet, because the bit length of the hexadecimal representation of large integers may vary. Actual length detection and signature processing is not complicated, but the TinyECC library is very selective on input parameters and requires accurate length information before a signature validation. As a consequence, the signature in HS2 is encoded in the Abstract Syntax Notation One (ANS.1), inherently including length information for *r* and *s*. [55]

The reply can be sent with two plain fixed-length arrays for the signatures, because the powerful gateway can handle necessary computations to strip any padding and to encode the signature correctly. Thus, the computation time on the node is minimized. Similarly to the three handshake messages, the two necessary authentication messages of the aggregator to the collectors are implemented (cf. Figure 3). The

aggregator's verification message 1 (AV1), send from the collector to the gateway upon receipt of the aggregator announcement, includes *msgType*, *hsType*, and the aggregator *address*. Additionally, AV2, send from the gateway to the collector, includes the *public agg x*, *public agg y*, and a *signature* from the previously authenticated gateway, confirming the aggregator's public key with the given address. [55]

```
// Handshake Message 2 (HS2):
typedef nx_struct to_hsm2_t{
   nx_uint8_t msgType;
   nx_uint8_t hsType; // 0x02
   nx_uint8_t public_x[24];
   nx_uint8_t public_y[24];
   nx_uint8_t digest[20];
   //variable length, ANS.1 encoded
   nx_uint8_t signature[];
} to_hsm2_t;
```

Listing 1: Example for handshake message.

1.6 Evaluation

TinyTO is evaluated for memory and energy consumption and its ability to fit those requirements of class 1 devices. Furthermore, the performance is analyzed and the security level is compared to related work. In order to show the feasibility of TinyTO for class 1 devices, TelosB nodes are used exclusively in a test-bed for evaluation purposes. TinyOS is the operating system chosen for the current implementation. A simplified setup for the test-bed is shown in Figure 1 and the following situation is assumed: In rooms A, C, and D the light is turned off and the room temperature is low. Sensors in room B report lights being switched on and the microphone shows a high noise level. All data collected is send either directly (see room B – use-case 1) or via multiple hops (see rooms A, C, and D – use-case 2) to the gateway. Software on the gateway can analyze the data collected and sends corresponding information to other systems (*e.g.*, climate control

or room booking system). The analysis result for room B indicates that a conference takes place and, thus, the climate control is activated and an entry is made in the room's calendar that room B is currently occupied. For rooms A, C, and D the internal room lock system is informed that these rooms are empty and shall be locked automatically. In this example the addressing of inconspicuous data can lead to the claim that confidential information collected allows for conclusions about room occupancy. Additionally, this introduces security risks in the application, since in case of room information being retrieved by eavesdropping due to missing security in the communication of those sensors; an attacker can become aware of this situation and could plan for a burglary.

Memory Consumption

TinyTO's main challenge was to require only a small part of the resources available, allowing applications (*e.g.*, TinyIPFIX [9]) to run in addition to the security solution. The memory consumption of applications can be determined for TinyOS directly from the compiling tool, because resources are already known at the time of compilation. Deactivation of components (*e.g.*, RPL or TinyECC optimizations) via the compiling tool or removing components (*e.g.*, TinyTO component or HandshakeHandler) in the code allows for a recording of the individual memory consumption of TinyTO components shown in Table 2. The small memory difference between conceptually identical components (handshake and cryptography) in collector's and aggregator's implementation originates from marginally different use-cases as described before and illustrated in Figure 1. For example, collectors only need to store one message at a time, but aggregators need additional memory to buffer data before that aggregation can be performed [9]. Since the memory is statically reserved, the detailed memory footprint

depends on the degree of aggregation (*doa*). Furthermore, collectors only need code for an encryption, whereas aggregators need code for the decryption and the encryption. In comparison to aggregators, collectors periodically read their sensor values, which requires additional memory. Overall, this leads to a ROM consumption of 37,590 Byte purely for collector and 33,174 Byte for aggregator applications (including data handling [9] and RPL [60, 41]), as shown in Table 2, and yields a slightly more than 4 kByte of additional free memory for aggregators, which can be used to enable ECC optimizations. Table 3 shows that optimizations have a direct impact on memory consumption and if they are used (indicated by X) on TinyTO.

Table 2: Memory consumption of components [55].

Operation	**Aggregator**		**Collector**	
	ROM [Byte]	**RAM [Byte]**	**ROM [Byte]**	**RAM [Byte]**
Handshake	1636	602	1138	612
Cryptography	11406	406	9378	406
TinyTO total	**13042**	**1018**	**10516**	**1018**
Data Handling [9]	26904	6964	31144	5478
RPL [60, 41]	6270	498	6228	1498
Total	**46216**	**8470**	**48114**	**7994**

Table 3: Memory consumption of TinyECC optimizations [55].

Operation	**ROM [Byte]**	**RAM [Byte]**	**Aggregator**	**Collector**
Barrett reduction	780	114	-	-
Hybrid multiplication	12	0	X	-
Hybrid square	114	0	X	X
Secpt optimization	414	0	X	-
Projective coordinates	850	0	X	X
Sliding window	206	2350	-	-

Run Time Performance

In terms of performance, the slow microcontroller and limited memory have a high impact on all results. A message size of 116 Byte was assumed, because it is typically

used for the application TinyIPFIX supported [9]. Further collectors do not need to decrypt data. For measurements performed a timer was read before and after an operation was executed. The resolution was at 65.53 ms, allowing accurate measurements in the scope of several seconds. Table 4 shows the execution times for various cryptographic operations in TinyTO's aggregators and collectors. Aggregators are generally almost twice as fast as collectors, which is mainly due to more activated ECC optimizations (cf. Table 3). Liu et al. give a performance evaluation for TelosB showing the speed for ECC operations, when all optimizations are used [16]: ECDSA signing takes only about 1.6 s (TinyTO aggregator needs about 5.14 s, TinyTO collector about 9.28 s), verification about 2 s (TinyTO aggregator needs around 10.20 s, TinyTO collector around 18.51 s). This is much faster than TinyTO and proves the performance limitations due to the restricted memory of chosen hardware (here: TelosB).

Table 4: Execution times for ECC operations [55].

ECC Operation	**Aggregator [s]**	**Collector [s]**
EC Key Generation	4.77	8.77
SHA-1	≤ 0.10	≤ 0.10
ECDSA Sign	5.14	9.28
ECDSA Verify	10.20	18.51
ECIES Encrypt	5.98	9.41
ECIES Decrypt	4.96	-

Table 5 shows the execution times of composite operations, including transmission times in both directions and response calculations times. The fact that the gateway performs faster than nodes has no influence on the overall result. The value for a message aggregation with *doa* = 2 is calculated based on two decryption and one encryption operation on the aggregator.

Table 5: Energy consumption of composite operations [55].

Operation	Time [s]	Energy [mJ]
Handshake Aggregator	20.14	85.90
Handshake Collector	36.59	154.99
Aggregator verification	18.52	78.58
Message aggregation (*doa* = 2)	15.90	68.28

Based on those results, it is possible to determine the minimal interval t, where collectors send their encrypted messages and aggregators can still catch up with incoming packets: Once the handshake is executed, collectors send data after t = 9.41 s, which is the minimal time needed for encryption, even if data is immediately available. Assuming, sufficient memory on an aggregator‘s device is available for caching packets of degree *doa*, plus an aggregate computation, an aggregator requires $t = doa * 4.96$ s + 5.89 s to decrypt incoming messages of the degree *doa* and to encrypt the aggregate. For *doa* = 2 the aggregator needs t = 15.81 s. This equation does hold for a small degree of aggregation and small networks (*e.g.*, 20 nodes). The formula for t must be adapted with the required time for ECIES encryption and decryption, if the aggregator (a) is more powerful, (b) can support a greater degree of aggregation, and (c) can perform faster operations and if the network becomes larger.

Energy Consumption

WSN devices are usually battery-powered and depend on the deployment, which makes an exchange not really easily. Hence, TinyTO must be energy-efficient to avoid a fast battery depletion. TelosB nodes in the test-bed are powered by two off-the-shelf batteries, each with a capacity of 2,000 mAh and voltage of 1.5 V, in total delivering U = 3.0 V. Wireless data transmissions and computations in the micro-controller show the biggest

impact on energy consumption. Auxiliary components, such as LEDs or serial connectors, are not taken into consideration, as they typically are deactivated during a final deployment.

TelosB nodes are equipped with a CC2420 RF chip on the IEEE 802.15.4 2.4 GHz band [48], which has a maximum output power of -25 dBm to 0 dBm for data transmissions [10]. Assuming 0 dBm, the current draw for sending (*Tx*) is at I_{Tx} = 17.4 mA and at I_{Rx} = 19.7 mA for receiving (*Rx*) [49] and [50]. The theoretical transmission rate of the CC2420 is at 250 kbps, but some practical measurements are as low as 180 kbps. For the purposes of the calculation being as close to reality as possible, it can be assumed that the full transmission rate *R* is never actually reached and $R = 220 \pm 20$ kbps. Knowing the transmission rate and implementation details of messages (cf. Section 1.5), allows for the calculation of the transmission time for each message and the total energy consumption *ER* as follows: *ER* depends on the message size *S* for a given voltage, a current draw, and a transmission rate, resulting in $ER(S) = U * I * S$. In particular this concerns the data handling with TinyIPFIX [9], the three handshake messages (HS1 to HS3), and the aggregator verification (AV1 and AV2) [55]. In comparison to ECC operations TinyIPFIX operations are almost instantaneous and negligible in the light of the overall energy consumption of TinyTO. Furthermore, messages as outlines in the handshake design consider only the size of individual data fields in an unencrypted message (cf. Section 1.5). In reality, the packets transmitted are much larger than supported by IEEE 802.15.4 on the MAC layer (102 Byte out of the total frame size of 127 Byte [48]). Hence, packet fragmentation support for TinyTO is essential. Because 12 Byte are used by TinyOS and the cyclic redundancy check (CRC)

for error-detection is added, 90 Byte remain for the actual payload in every message on the MAC layer.

Table 6: Energy consumption of the radio transmission [55].

Message	ps [Byte]	DS [Byte]	Time [ms]	Energy [mJ]
HS 1 (Tx)	139	223	8.11	0.42
HS 2 (Tx)	189	300	10.91	0.64
HS 3 (Tx)	114	203	7.38	0.38
AV 1 (Tx)	82	171	6.22	0.32
AV 2 (Tx)	168	242	8.80	0.52

TinyTO handshake messages are not transmitted as plaintext, but in larger Elliptic Curve Integrated Encryption Scheme (ECIES) cipher texts, requiring 69 Byte (1 Byte to indicate the point compression type, 24 Byte for each Elliptic Curve (EC) point component, and 20 Byte for the message authentication more than the pure message size). For example, a HS1 message has a size of 70 Byte and a size of 139 Byte after encryption with ECIES. Given the maximum payload size of 90 Byte, HS1 is fragmented to fit into MAC layer packets. Every fragment requires additional headers and other fields (*e.g.*, fragment number and header field indicating fragmentation). Thus, the effective data size *DS*, which is transmitted to convey a payload of size *ps* = 139 Byte, is calculated as $DS = ps + \lceil ps \,/\, 90\ \text{Byte} \rceil * 37\ \text{Byte} = 213\ \text{Byte}$. Table 6 shows the results of these considerations for different message types and the energy consumption for their transmissions. It can be stated that the energy consumption for HS2 is the highest with 6.34 mJ compared to HS1 and HS3 due to its enormous message size of 189 Byte. If a message size is approximately 100 Byte the energy consumption levels around 0.3 mJ.

Similarly to energy calculations for radio transmissions, the energy consumption of the microcontroller (MSP430F1611 16 bit Ultra-Low-Power Micro-Controller Unit

(MCU) from Texas Instruments [51]) for different cryptographic operations can be calculated. The current draw in active mode (*i.e.* only MCU and no radio transmissions) is given as 1.8 mA [52] and [49]. However, experimental measurements show that the relevant difference between idle and a fully utilized MCU is only at I_{AM} = 1.4 mA. The formula used to calculate the energy consumption E_{MCU} of the MCU depending on the computation time t subsequently is $E_{MCU}(t) = U * I_{AM} * t$. As shown in Table 7 the energy consumption differs for the aggregator and the collector due to different activations of these ECC optimizations. Given the cost of a radio transmission (cf. Table 6) and the computation of single cryptographic operations (cf. Table 7) the energy consumption for the entire handshake and similarly more complex sequences of operations can be calculated. According to the design shown in Figure 2, the handshake requires six operations: EC Key Generation, sending of HS 1, reception of HS 2, ECDSA signature verification, ECDSA signature signing, and sending of HS 3. Similarly, the verification of an aggregator needs three operations (cf. Figure 3): Sending of the aggregator's identity in AV1, the reception of the signed message containing the public key in AV2, and a ECDSA signature verification. Aggregation with *doa* = 2 requires a combination of the reception of two data packets including data collected, two times an ECIES decryption, the ECIES encryption, and a sending of one aggregated data packet. Table 5 shows the corresponding times and energy consumptions.

Table 7: Energy consumption of cryptographic operations [55].

Operation	**Aggregator**		**Collector**	
	Time [s]	**Energy [mJ]**	**Time [s]**	**Energy [mJ]**
EC Key Generation	4.77	20.03	8.77	36.83
ECDSA Sign	5.14	21.59	9.28	38.98
ECDSA Verify	10.20	42.84	18.51	77.74
ECIES Encrypt	5.98	25.12	9.41	39.52
ECIES Decrypt	4.96	20.83	-	-

The battery powered TelosB requires a minimal voltage of 1.8 V [10], meaning a battery cannot be depleted to an energy level below 60% of the original charge; otherwise the voltage will drop below that threshold. Thus, it can be calculated that 12.96 kJ are available in one set of batteries. Measurements show that TelosB nodes draw on average of 70.7 mA, while remaining idle when no sleep modes for MCU and radio are activated. Thus, the expected runtime without any application is about 12.96 kJ = 61,103 s, or roughly 16 hours and 58 min for one 3V $*$ 70.7 mA set of batteries. If collectors are programmed to collect, encrypt, and send data in the format proposed by [9], every interval t the impact on the runtime of collectors and their aggregators can be calculated accordingly. Assuming every tenth transmission contains the TinyIPFIX template (only meta information) instead of a TinyIPFIX record (data values) and an aggregation with *doa* = 2 is performed [9], the same batteries will last for 16 hours and 53 min in aggregators (which compares to a reduction of 0.5% or 5 min) and 16 hours and 55 min in collectors (reduction of 0.3% or 3 min), given $t = 1\ \text{min}^{-1}$.

1.7 Summary

In this book chapter a new handshake for two-way authentication and key-exchange was introduced, which provides end-to-end security for class 1 devices in the IoT domain. The newly developed protocol TinyTO is based on the Bellare-Canetti-Krawczyk protocol with an additional PSK extension for secure authentication. In order to match major resource constraints, TinyTO applies energy-efficient ECC operations for cryptographic functions and uses pre-shared master keys (with a length of 16 Byte) for an

authentication toward the gateway only. Furthermore, TinyTO supports secure data aggregations with a small overhead, which is the key for today's IoT applications. Transferring TinyTO to more resourceful devices will enhance performance, because additional ECC optimizations can be activated and the responsiveness of the network will increase. Finally, the maximum degree of aggregations can be increased in these cases, since more memory for buffering data will be available.

Acknowledgements

This work was supported partially by the FLAMINGO [53] and the SmartenIT [54] projects, funded by the EU FP7 Program under Contract No. FP7-2012-ICT-318488 and No. FP7-2012-ICT-317846, respectively.

References

[1] L. Atzori, A. Iera, and G. Morabito: *The Internet of Things: A Survey*, Computer Networks, ELSEVIER, Atlanta, GA, U.S.A., Vol. 54, No. 15, pp 2787–2805, October 2010.

[2] C. Alcaraz, P. Najera, J. Lopez, and R. Roman: *Wireless Sensor Networks and the Internet of Things: Do We Need a Complete Integration?*, 1st International Workshop on the Security of the Internet of Things (SecIoT), Tokyo, Japan , pp 1–8, December 2010.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci: *Wireless Sensor Networks: A Survey*, Computer Networks, ELSEVIER, Atlanta, GA, U.S.A., Vol. 38, No. 4, pp. 393–422, March 2002.

[4] A. Perrig, J. Stankovic, and D. Wagner: *Security in Wireless Sensor Networks*, Communications of the ACM, New York, NY, U.S.A.,Vol. 47, No. 6, pp. 53–57, June 2004.

[5] S. Hausmann: *Internet of Things a Risk-Reward Proposition For Security Professionals*, securityinfowatch. [Online]. Available: http://www.securityinfowatch.com/article/11714106/navigating-security-threats-posed-by-internet-of-things-technology, October 7, 2014.

[6] R. H. Weber: *Internet of Things - New Security and Privacy Challenges, Computer Law and Security Review*, ELSEVIER, Atlanta, GA, U.S.A., Vol. 26, No. 1, pp 23–30, March 2010.

[7] C. Medaglia and A. Serbanati: *An Overview of Privacy and Security Issues in the Internet of Things*, The Internet of Things, D. Giusto, A. Iera, G. Morabito, and L. Atzori (Eds.), Springer, New York, NY, U.S.A., pp 389–395, January 2010.

[8] C. Bormann, M. Ersue, and A. Keranen: *Terminology for Constrained-Node Networks*, RFC 7228, IETF, Internet Engineering Task Force, Fermont, CA, U.S.A., [Online]. Available: http://www.ietf.org/rfc/rfc7228.txt, May 2014.

[9] C. Schmitt, T. Kothmayr, B. Ertl, W. Hu, L. Braun, and G. Carle: *TinyIPFIX: An Efficient Application Protocol For Data Exchange Cyber Physical Systems*, Journal of Computer Communications, ELSEVIER, Atlanta, GA, U.S.A., Vol. 56, No. 2, pp 257–268, June 2014.

[10] Advantic Sistemas y Servicios S.L.: *TelosB CM5000-SMA*, [Online]. Available: http://www.advanticsys.com/shop/ mtmcm5000sma-p-23.html, November 2015.

[11] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle: *DTLS-based Security and Two-way Authentication for the Internet of Things*, Ad Hoc Networks, ELSEVIER, Atlanta, GA, U.S.A., Vol. 11, No. 8, pp 2710–2723, November 2013.

[12] W. Du, R. Wang, and P. Ning: *An Efficient Scheme for Authenticating Public Keys in Sensor Networks*, 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Urbana-Champaign, IL, U.S.A. pp 58–67, May 2005.

[13] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine: *Authentic Third-Party Data Publication,* Data and Application Security, B. Thuraisingham, R. van de Riet, K. Dittrich, and Z. Tari, (Eds.) Springer, New York, NY, U.S.A., Vol. 73, pp 101–112, 2001.

[14] Q. Chang, Y.-P. Zhang, and L.-L. Qin: *A Node Authentication Protocol Based on ECC in WSN*, International Conference on Computer Design and Applications (ICCDA), Qinhuangdao, Hebei, China, pp 606–609, June 2010.

[15] Y.-S. Jeong and S.-H. Lee: *Hybrid Key Establishment Protocol Based on ECC for Wireless Sensor Network*, Ubiquitous Intelligence and Computing (UIC), LNCS, Vol. 4611, Springer, Heidelberg, Germany, pp 1233–1242, June 2007.

[16] Y. Liu, J. Li, and M. Guizani: *PKC Based Broadcast Authentication using Signature Amortization for WSNs*, IEEE Transactions on Wireless Communications, New York, NY, U.S.A., Vol. 11, No. 6, pp 2106–2115, June 2012.

[17] E. B. Barker, W. C. Barker, W. E. Burr, W. T. Polk, and M. E. Smid: *SP 800-57. Recommendation for Key Management, Part 1: General (Revised)*, National Institute of Standards & Technology (NIST), Gaithersburg, MD, U.S.A., 2007.

[18] P. Nie, J. Vähä-Herttua, T. Aura, and A. Gurtov: *Performance Analysis of HIP Diet Exchange for WSN Security Establishment*, 7th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet), Miami, FL, U.S.A., pp 51–56, October/November 2011.

[19] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, and M. Ylianttila, PAuthKey: *A Pervasive Authentication Protocol and Key Establishment Scheme for Wireless Sensor Networks in Distributed IoT Applications*, International Journal of Distributed Sensor Networks, hindawi Publishing Corporation, New York, NY, U.S.A., Vol. 2014, pp. 1– 14, July 2014.

[20] D. Westhoff, J. Girao, and A. Sarma: *Security Solutions for Wireless Sensor Networks*, NEC Journal of Advanced Technology, NEC, Tokyo, Japan, Vol. 59, No. 2, 2006.

[21] A. Weimerskirch and D. Westhoff: *Zero Common-Knowledge Authentication for Pervasive Networks*, Selected Areas in Cryptography, LNCS, Vol. 3006, Springer, Heidelberg, Germany, pp 73–87, August, 2004.

[22] Z. Shelby, K. Hartke, and Bormann: *The Constraint Application Protocol (CoAP)*, RFC 7252, IETF, Internet Engineering Task Force, Fermont, CA, U.S.A., [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt, June, 2014.

[23] M. Saleh and I. Al Khatib: *Throughput Analysis of WEP Security in Ad Hoc Sensor Networks*, 2nd International Conference on Innovations in Information Technology, Dubai, United Arab Emirates, pp 26–28, September 2005.

[24] W. Hu, H. Tan, P. Corke, W. Shih, and S. Jha: *Toward Trusted Wireless Sensor Networks*, Transactions on Sensor Networks, ACM, New York, NY, U.S.A., Vol. 7, No. 1, pp 5:1–5:25, August 2010.

[25] J. Linn: *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*, RFC 1421, IETF, Internet Engineering Task Force, Fermont, CA, U.S.A., [Online]. Available: http://www.ietf.org/rfc/rfc1421.txt, February 1993.

[26] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. Shantz: *Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks*, Third International Conference on Pervasive Computing and Communications, New York, NY, U.S.A., pp 324–328, March 2005.

[27] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz: *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*, Cryptographic Hardware and Embedded Systems, , LNCS, Vol. 3156, Springer, Heidelberg, Germany, pp 119–132, 2004.

[28] D. Xiaojiang and C. Hsiao-Hwa: *Security in Wireless Sensor Networks*, IEEE Wireless Communications, New York, NY, U.S.A., Vol. 15, No. 4, pp 60–66, August 2008.

[29] C. Karlof and D. Wagner: *Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures*, Ad Hoc Networks, ELSEVIER, Atlanta, GA, U.S.A., Vol. 1, No. 2, pp 293– 315, 2003.

[30] G. Stoneburner: *SP 800-33: Underlying Technical Models for Information Technology Security*, Tech. Rep., National Institute of Standards and Technology (NIST), Washington, DC, U.S.A. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-33/sp800-33.pdf, December 2001.

[31] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz: *A Comparison Of Mechanisms For Improving TCP Performance Over Wireless Links*, IEEE/ACM Transactions on Networking, New York, NY, U.S.A., Vol. 5, No. 6, pp 756– 769, December 1997.

[32] A. Menezes, P. Van Oorschot, and S. Vanstone: *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, U.S.A., October 2010.

[33] W. Diffie, P. Van Oorschot, and M. Wiener: *Authentication and Authenticated Key Exchanges*, Designs, Codes and Cryptography, Springer, New York, NY, U.S.A., Vol. 2, No. 2, pp 107–125, March 1992.

[34] R. Canetti and H. Krawczyk: *Analysis of Key-exchange Protocols and Their Use for Building Secure Channels*, Advances in Cryptology – EUROCRYPT, LNCS, Vol. 2139, Springer, Heidelberg, Germany, pp 453–474, April 2001.

[35] H. Delfs and H. Knebl: *Introduction to Cryptography: Principles and Applications*, Information Security and Cryptography, Springer, Heidelberg, Germany, May 2007.

[36] M. Bellare, R. Canetti, and H. Krawczyk: *A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract)*, 13th Annual ACM Symposium on Theory of Computing (STOC), Dallas, TX, U.S.A., pp 419–428, May 1998.

[37] S. Blake-Wilson and A. Menezes: *Authenticated Diffie-Hellman Key Agreement Protocols*, Selected Areas in Cryptography (SAC), Springer, London, U.K., pp 339–361, August 1999.

[38] B. LaMacchia, K. Lauter, and A. Mityagin: *Stronger Security of Authenticated Key Exchange*, Provable Security, LNCS, Vol. 4784, Springer, Heidelberg, Germany, Vol. 4784, pp 1–16, 2007.

[39] S. Blake-Wilson, D. Johnson, and A. Menezes: *Key Agreement Protocols and Their Security Analysis*, Cryptography and Coding. LNCS, Vol. 1355, Springer, Heidelberg, Germany, pp 30–45, December 1997.

[40] C. Boyd and A. Mathuria: *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, Springer, Berlin Heidelberg, Germany, December 2010.

[41] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.P. Vasseur, and R. Alexander: *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, RFC 6550, IETF Internet Engineering Task Force, Fermont, CA, U.S.A., [Online] https://tools.ietf.org/html/rfc6550, March 2012.

[42] D. Basin, C. Cremers, and S. Meier: *Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication*, Principles of Security and Trust, LNCS, Vol. 7215, Springer, Heidelberg, Germany, pp 129–148, 2012.

[43] S. Boeyen, T. Howes, and P. Richard: *Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2*, RFC 2559, IETF, Internet Engineering Task Force, Fermont, CA, U.S.A., [Online]. Available: http://www.ietf.org/rfc/rfc2559.txt, April 1999.

[44] M. Badra and I. Hajjeh: *Key-exchange Authentication Using Shared Secrets*, IEEE Computer Journal, New York, NY, U.S.A., Vol. 39, No. 3, pp 58–66, March 2006.

[45] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart: *HTTP Authentication: Basic and Digest Access Authentication*, RFC 2617, IETF, Internet Engineering Task Force, Fermont, CA, U.S.A., [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt, June 1999.

[46] B. Preneel: *Analysis and Design of Cryptographic Hash Functions*, Ph.D. Thesis, KU Leuven, Netherlands, [Online]. Available: http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf, February 1993.

[47] P. Rogaway and T. Shrimpton: *Cryptographic Hash-function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second Preimage Resistance, and Collision Resistance*, Fast Software Encryption, LNCS, Vol. 3329, Springer, Heidelberg, Germany, pp 371–388, June 2004.

[48] Texas Instruments: *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf, November 2015.

[49] H. A. Nguyen, A. Forster, D. Puccinelli, and S. Giordano: *Sensor Node Lifetime: An Experimental Study*, IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM), New York, NY, U.S.A., pp 202–207, March 2011.

[50] C. Sadler and M. Martonosi: *Data Compression Algorithms for Energy-constrained Devices in Delay Tolerant Networks*, 4th International Conference on Embedded Networked Sensor Systems (SenSys), ACM, New York, NY, U.S.A., pp 265–278, October/November 2006.

[51] Texas Instruments: *MSP430F161x Mixed Signal Microcontroller Datasheet*, http://www.ti.com/lit/ds/symlink/msp430f1611.pdf, November 2015.

[52] J. Polastre, R. Szewczyk, and D. Culler: *Telos: Enabling Ultra-low Power Wireless Research*, 4th International Symposium on Information Processing in Sensor Networks (IPSN), IEEE Press, Piscataway, NJ, U.S.A., pp 364–369, April 2005.

[53] FLAMINGO Consortium: *FLAMINGO: Management of the Future Internet*, [Online]. Available: http://www.fp7-flamingo.eu/, November 2015.

[54] SmartenIT Consortium: *SmartenIT: Socially-aware Management of New Overlay Application Traffic combined with Energy Efficiency in the Internet*, [Online]. Available: http://www.smartenit.eu/, November 2015.

[55] M. Noack: *Optimization of Two-way Authentication Protocol in Internet of Things*, Master Thesis, Universität Zürich, Communication Systems Group, Department of Informatics, Zürich, Switzerland, [Online] https://files.ifi.uzh.ch/CSG/staff/schmitt/Extern/Theses/Martin_Noack_MA.pdf, August 2014.

[56] E. Rescorla: *SSL and TLS: Building and Designing Secure Systems*, Addison-Wesley Longman, Amsterdam, Netherlands, October 2000.

[57] K. Schmeh: *Kryptografie: Verfahren – Protokolle – Infrastrukturen*, dpunkt.verlag GmbH, Heidelberg, Germany, Vol. 5, February 2013.

[58] J. Katz and Y. Lindell: *Introduction to Modern Cryptography*, CRC Press, Boca Raton, FL, U.S.A., Vol. 2, November 2014.

[59] TinyOS: *BLIP Tutorial*, [Online] http://tinyos.stanford.edu/tinyos-wiki/index.php/BLIP_Tutorial, November 2015.

[60] J.G. Ko, S. Dawson-Haggerty, D.E. Culler, J.W. Hui, and P. Levis: *Connecting Low-Power and Lossy Networks to the Internet*, IEEE Communications Magazine, New York, NY, U.S.A., Vol. 49, No. 4, pp 96-101, April 2011.