Flexible Key Exchange Negotiation for Wireless Sensor Networks

Giuseppe Bianchi
Department of Electronic Engineering
University of Rome 2 “Tor Vergata”, Italy
giuseppe.bianchi@uniroma2.it

Angelo T. Capossele, Alessandro Mei, Chiara Petrioli
Department of Computer Science
“Sapienza” University of Rome, Italy
{capossele, mei, petrioli}@di.uniroma1.it

ABSTRACT

Despite recent improvements of the capabilities of Wireless Sensor Networks (WSN) nodes, network protocol support for key management is still lagging behind. While in traditional networks well known protocol suites (e.g., IPsec IKE and the TLS handshake), are commonly used for flexible negotiation of the cryptographic and key exchange protocols, to the best of our knowledge no similar support has been provided for the same operation in WSNs. The goal of this paper is therefore threefold. We discuss the design of a flexible security negotiation protocol for WSNs, and we suggest to adapt TLS handshake ideas to obtain maximum flexibility. We design and implement a security association set up protocol, tailored to the resource constraints and limits of WSN nodes. Finally, we run an experimental assessment of this protocol operations in support of RSA key transport, Elliptic Curve Diffie-Hellman key agreement, and Identity Based Encryption key agreement.

Categories and Subject Descriptors

D.4.4 [**Communications Management**]: Network communication; D.4.6 [**Security and Protection**]: Cryptographic controls.

General Terms

Security, Design.

Keywords

Sensor network security, secure communication architecture, identity based encryption.

1. INTRODUCTION

Application scenarios for wireless sensor networks (WSNs) are extremely diverse and heterogeneous [1, 2], ranging from smart environment to perimeter sensing, to weather and ambient control, to healthcare, to military applications, and so on. With so much diversity, a one-size-fits-all general design paradigm for WSN security appears far from being effective, if even possible.

Even restricting (as in this paper) to point-to-point unicast communication protection, only a careful analysis of the specific requirements emerging in a given scenario may tell whether protection should take place at link layer, for hop-by-hop communications, or at network/transport layer, for end-to-end communications, or at both layers. Similarly, a “better than nothing” level of protection may fulfill the needs of some deployments, but may turn out to be largely insufficient in hostile environments or in mission critical scenarios such as military or healthcare ones. Finally, the huge heterogeneity in the sensor nodes capability (in terms of memory, computational, or energy requirements) further plays against the identification of a “unique” or “common” security solution set, whereas they call for a large spectrum of security level versus resource consumption trade-offs.

The above discussion underlines the need for the research community not only to limit its investigation on the identification and design of new security protocols or suites, but also to complementarily address approaches devised to permit *flexibility and extensibility* in the choice (i.e., negotiation) of which specific algorithm or cipher suite fits the actual need of a target WSN deployment.

The aim of this paper is to test, through implementation and experimental assessment, design ideas for the specification of a security protocol for the establishment of (dynamic) security associations across pairs of possibly heterogeneous WSN nodes. We specifically focus on the case of unicast associations. Albeit restrictive and not sufficient to cover the variety of WSN communication security needs, the design of a WSN unicast security protocol nevertheless represents a first step which can be exploited in a variety of scenarios. Moreover, the problem of unicast security has been extensively addressed in traditional networks throughout several years of evolution, yielding extremely mature security association management protocols such as the Internet Key Exchange (IKE) protocol [3] employed in the IPsec framework, or the handshake phase of the Transport Layer Security (TLS) protocol [4], proven adaptable to contexts (e.g., layer 2 authentication with EAP-TLS/TTLS) other than web security for which it was originally conceived. However, extending these approaches to resource-constrained wireless nodes requires careful investigation and is the subject of this work. More specifically, we make the following contributions to flexible security provisioning for WSNs.

- We define a new protocol for key management and cipher suite negotiation. The protocol draws several basic concepts from the TLS handshake, because of its capability to support the negotiation of the cryptographic key management approach itself in *addition* to the negotiation of symmetric cipher suites for message authentication and encryption.

- To demonstrate the viability of our design, we implement it on TelosB and MICA2 sensor nodes. Our proof-of-concept implementation supports three different key management mechanisms: A key transport and a key agreement schemes based on traditional RSA and Elliptic Curve Cryptography (ECC), respectively, and a key agreement approach based on Identity Based Cryptography (IBC) [5].
- We experimentally assess the performance of our implementation vs. message and energy consumption overhead, showing that the extra complexity and overhead posed by our proposed negotiation scheme is minimal.

The paper is organized as follows. In the next section we review works on security for WSNs. Section 3 provides the necessary background and considerations for the definition of our negotiation protocol, then described in Section 4. In Section 5 we evaluate the costs of our mechanism associated to RSA, ECC and ID-NIKDS, a dialect of IBC. Section 6 concludes the paper.

2. RELATED WORKS

Security in WSNs has been extensively addressed, both in terms of message encryption and integrity, as well as node authentication and key exchange, including the support of public key based approaches which is considered viable since at least 6 years [6, 7] even on severely resource constrained sensor nodes.

WSN security frameworks supporting symmetric encryption and message authentication have been largely deployed, the most known perhaps being TinySec [8] and MiniSec [9]. The former is the first fully implemented protocol for link layer cryptography in WSNs. It provides security services similar to IPsec (header authentication and authenticated encryption). It achieves low energy consumption and memory usage through tailored optimizations (e.g. reusing part of the packet header as Initialization Vector) and weakened security (e.g. no protection against replay attacks). MiniSec deploys security at network layer, for both unicast and broadcast communications, and improves the protection level at the expense of a slightly higher sensor node resource commitment.

However, neither TinySec nor MiniSec are integrated in a comprehensive security architecture that includes a specifically devised protocol for key management support. These frameworks are thought for static environments: Security services must be set at compilation time, and use a pre-established cryptographic key, which makes the whole network insecure even by compromising a single node.

To the best of our knowledge, Sizzle [10] and Tiny-3TLS [11] are the only works that have addressed the design of a security architecture for WSNs. Both these works, however, focus on the porting of the TLS protocol *as is* over WSNs, and specifically on the development of secure web servers over sensor nodes and on the specification of gateway functionalities for providing end-to-end security and secure querying from "legacy" clients to sensor nodes. TLS is extended to support Elliptic Curve Cryptography [12] to make it viable in a WSN environment. Unlike these works, we do not promote the porting of TLS to the WSN environment. Rather, we inherit TLS cipher suite negotiation ideas applying them to a protocol purely dedicated to key exchange and security association set-up (i.e., a protocol whose goals are similar to those of IPsec IKE), and not necessarily meant to operate on a end-to-end basis (although, of course, it could).

3. BACKGROUND

3.1 IPsec IKE and TLS

Widely employed standard-based security association/session set up protocols, such as the IPsec Internet Key Exchange (IKE) protocol [3], or the handshake phase of the Transport Layer Security (TLS) protocol [4] and its UDP-based version (DTLS) [13], do not restrict their operations to the exchange of cryptographic information for symmetric key derivation. Rather, they further permit to flexibly *negotiate* the specific security services and relevant cipher suites to be employed.

Roughly speaking, the peer starting the handshake informs the other peer about the supported or desired cipher suites. The responding peer commits the decision of which specific cryptographic algorithms, among the offered ones, shall be employed. The handshake then proceeds with the usual exchange of information needed to establish a common cryptographic secret, from which detailed per-session keys are derived. The session set up is concluded by authenticating the previous information exchange and cipher suite negotiation phase, thus preventing attacks to the negotiation protocol.

Besides technical details, the most notable difference between IPsec IKE and the (D)TLS handshake resides in the fact that IPsec IKE assumes Diffie-Hellman as key agreement mechanism and only permits negotiation of the authentication and encryption algorithms, whereas (D)TLS has the further flexibility of permitting to choose which specific approach shall be employed for cryptographic key management (e.g., RSA-based key transport, Diffie-Hellman Key agreement in one of its variants—Anonymous, Fixed, Ephemeral—, etc). As such, an approach inspired to the TLS handshake appears more exploitable in the WSN environment.

For the convenience of the reader, we now briefly review the TSL negotiation and key management operation. In TLS, the handshake starts with a client sending an CLIENTHELLO message to the server. This message carries three kinds of information: Protocol-specific information (e.g., SSL/TLS version used, length, session, compression algorithms supported, etc.), a 32B nonce made up of 28B random number plus 4B timestamp, and a list of supported cipher suites. Each cipher suite specifies three[1] algorithms for key exchange, symmetric encryption, and message authentication, respectively. For instance, the cipher suite TLS_RSA_WITH _RC4_128_SHA implies key transport via RSA (Figure 1), RC4 symmetric encryption with 128b key to be used for the encrypted data transfer following the TLS handshake phase,

[1] Since TLS version 1.2 [4], a fourth algorithm for the Pseudo Random Function used in the subsequent key derivation phase may be optionally further specified.

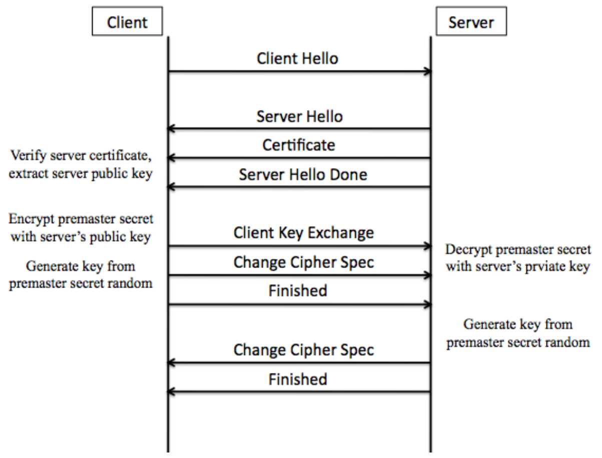
Client
Server
Client Hello
Server Hello
Certificate
Server Hello Done
Verify server certificate, extract server public key
Encrypt premaster secret with server's public key
Client Key Exchange
Change Cipher Spec
Finished
Decrypt premaster secret with server's prviate key
Generate key from premaster secret random
Generate key from premaster secret random
Change Cipher Spec
Finished

Figure 1: TLS key transport with RSA.

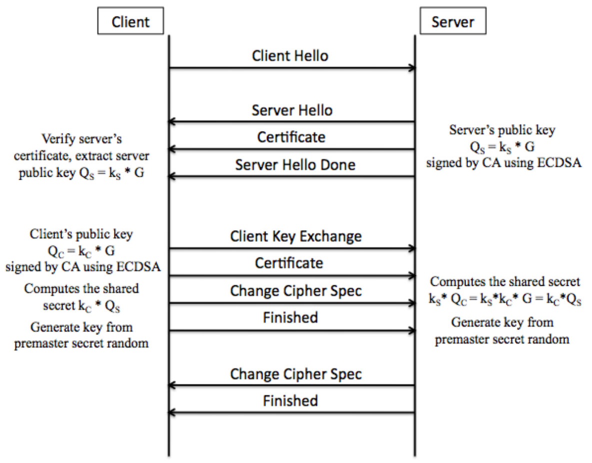
Client
Server
Client Hello
Server Hello
Certificate
Server Hello Done
Verify server's certificate, extract server public key $Q_S = k_S * G$
Server's public key $Q_S = k_S * G$ signed by CA using ECDSA
Client's public key $Q_C = k_C * G$ signed by CA using ECDSA
Client Key Exchange
Certificate
Change Cipher Spec
Finished
Computes the shared secret $k_C * Q_S$
Generate key from premaster secret random
Computes the shared secret $k_S* Q_C = k_S*k_C* G = k_C*Q_S$
Generate key from premaster secret random
Change Cipher Spec
Finished

Figure 2: TLS key agreement with Fixed Diffie Hellman over Elliptic Curves.

and message authentication via SHA-1. Alternatively, a cipher suite TLS_ECDH_ECDSA_WITH_... would mandate for a "Fixed" Elliptic Curve Diffie-Hellman key agreement, i.e., with public coefficients signed with Elliptic Curve DSA signatures (Figure 2) [12].

The server replies with a ServerHello message which contains the server nonce, the chosen cipher suite, and supplementary data such as the session identifier etc. After the ServerHello message, the TLS handshake proceeds differently, depending on the specific key exchange algorithm selected.

In case of RSA key transport (Figure 1), the server sends a Certificate message containing its RSA public key inside an X.509v3 Certificate signed by a Certification Authority (or a Certificate chain), and closes the Server hello phase with a ServerHelloDone message. Once the client successfully checks that the RSA certificate has a valid signature, and that the signing Certification Authority is trusted, it sends back a ClientKeyExchange message containing a 46B *Pre-Master Secret*, generated by the client, and encrypted using the RSA Public Key of the Server. As a result, both client and server can derive (using the exchanged nonces, through Pseudo Random Functions - refer to the standard [4] for details) the session key (Master Secret), the needed cryptographic keys for symmetric encryption and message authentication, and, when applicable, initialization vectors.

In case of ECDH, the messages sent by the Server depend on which specific ECDH key exchange algorithm has been selected. If "Fixed" ECDH_ECDSA (Figure 2), the Server sends a certificate containing the ECDH-capable public key signed by the Certification Authority with ECDSA. If an "Anonymous" ECDH exchange is selected, the ECDH public key is delivered in a message called ServerKeyExchange. If, finally, an "Ephemeral" ECDH key exchange is performed, say ECDHE_RSA, then the ECDH public key delivered in the ServerKeyExchange message is signed with RSA, whose Public Key Certificate is delivered in the Certificate message. Similarly, the Client reply depends on the key exchange algorithm chosen. For instance, in the "Fixed" ECDH case, the certified ECDH client public key (obviously a point belonging to the same Elliptic Curve used by the Server) is delivered in a Certificate message (Figure 2), while the most consuming exchange is the "Ephemeral" case where both a Certificate and a ClientKeyExchange message are required. The Pre-Master Secret is then derived as usual (Figure 2), and cryptographic keys are derived exactly as described above for RSA.

Independently of the specific key exchange algorithm employed, the TLS handshake concludes with a further exchange that is crucial to prevent attacks to the negotiation process. All the above described messages are plaintext, and as such can be trivially tampered through a man-in-the-middle (MITM) attack. Specifically, a MITM might attack the negotiation procedure deployed through the exchange of the plaintext ClientHello and ServerHello messages, by modifying the ClientHello message removing all but the weakest cipher suites in the list of offered ones, i.e., leaving only the cipher suites that the attacker could break. This is prevented by exchanging two further messages on both sides. A first 1b long message, ChangeCipherSpec, simply tells the other peer to switch to authenticated/encrypted mode using the negotiated algorithms. The message immediately following is called Finished. It contains a message authentication code (MAC) of the hash of all the previous sent and received handshake messages as seen by the peer. Specifically, a pseudo-random function (PRF) keyed with the master secret and expanded to at least 12B [4]. The other peer will decrypt the received Finished message, and will verify the hash and the related MAC before proceeding further with the actual TLS data transfer session.

3.2 Cryptographic Primitives

Public key cryptography is widely deployed to secure computer systems and networks. One of the most known and used asymmetric cryptosystem is RSA [14], whose security is based on the computational hardness of factoring large numbers and on the RSA problem (taking the e^{th} root modulo a composite number n). Other cryptographic schemes are recently used that are more suited for devices that are severely resource-constrained like wireless sensor nodes.

One example is Elliptic Curve Cryptography (ECC). A cryptosystem based on elliptic curves is defined by a finite field F_q, a small set of parameters that describe the elliptic curve E/F_q, a point $P \in E(F_q)$, and the order n of P.

These parameters are chosen in such a way that the elliptic curve discrete logarithm problem (ECDLP) cannot be solved by the adversary in reasonable time. In the suite of algorithms based on ECC we have both protocols for key exchange (ECDH) and protocols for digital signatures (ECDSA). ECC promises to deliver the same security of RSA with much shorter keys. A comparison between different cryptographic systems [15] shows that RSA with a key size of 622b offers the same security as ECC with a key only 105b long. More recently, NIST recommended an RSA key size of 3072b and an ECC key size of 256b to build cryptosystems that can be safe until the year 2030 [16]. Clearly, shorter keys are important in WSNs since they use less memory in the device and consume less energy for transmission when sent over the wireless channel.

Another important cryptographic scheme is Identity Based Cryptography (IBC). IBC is a public key cryptosystem where any string is a valid public key. In particular, the ID of the sensor can be the public key. With IBC, certificate management is greatly simplified: Two sensors can communicate securely by exchanging their IDs and not large certificates, with a clear advantage in terms of energy used to set up the secure communication channel.

The most efficient IBC schemes are based on bilinear pairings on elliptic curves. Let $E/\mathbb{F}_q$ be an elliptic curve over a finite field $\mathbb{F}_q$, $E(\mathbb{F}_q)$ the set of points of this curve, and $\#E(\mathbb{F}_q)$ the order of the group. Let n be a positive integer, $\mathbb{G}$ an additive group of order n with identity 0, and $\mathbb{G}_T$ a multiplicative group of order n with identity 1. A bilinear pairing is an efficiently computable and non-degenerative mapping such that $\forall P, Q \in \mathbb{G}$ and $\forall a, b \in \mathbb{Z}^*$,

$$e([a]P, [b]Q) = e(P, [b]Q)^a = e([a]P, Q)^b = e(P, Q)^{ab}.$$

This latter property is called *bilinearity*. In this work we use pairings of type 1 (as defined in [17]), that also have the following property: $e(P, Q) = e(Q, P)$.

In an IBC system anyone is able to generate a public key from a known identity by using the *master public key*. Each public key is associated with a secret key that the Trusted Authority (TA) can generate by using the *master secret key*. In our setting, sensors can be preloaded with the master public key and the private key associated to their identity. Once the network is deployed, the trusted authority does not need to be online, and sensors and other nodes can get the public key of each other just by exchanging their IDs, with a clear advantage in terms of energy that has to be consumed for setting up a confidential and authenticated communication channel. Adding more sensors later is made possible by assigning new IDs and corresponding private keys. Due to these advantages, IBC is more and more used in security protocols designed for wireless sensor networks [18].

4. OUR NEGOTIATION PROTOCOL

A key exchange protocol for WSNs should be defined with the following core requirements in mind.

- It should enable the negotiation of the specific key exchange protocol employed.
- it should be decoupled from the encrypted/authenticated message delivery, i.e., it should be security-association oriented rather than session-oriented.
- It should minimize message transmission overhead and energy consumption, and be optimized for sensor node support.

The first requirement suggests to devise a negotiation mechanism more flexible than that used by IPsec IKE (based on Diffie-Hellman) and, rather, similar to the TLS handshake protocol. In other words, we should prefer a key exchange protocol that provides the flexibility to permit the negotiation of the preferred or most appropriate public key algorithm for key transport or key agreement.

The second requirement implies protocol operations quite different from those of TLS (where the handshake is integral part of the encrypted/authenticated bulk data transfer session). It rather implies a functionally similar to that of IPsec IKE, whose goal is limited to the set up of a security association across a pair of nodes, whereas the task of delivering authenticated or protected data packets is demanded to one or more different dedicated protocols (e.g., the IPsec Encapsulated Security Payload or the IPsec Authentication Header). In other words, our proposed key exchange protocol should provide negotiation for cryptographic algorithms, as well as keying material, to *existing* data security protocol specific for sensor nodes, such as TinySec [8] or MiniSec [9].

Finally, the third requirement mandates the tailored design and optimization of the protocol messages for limited overhead and energy consumption. Schemes such as IBC are appealing candidates for key management in WSNs, as they permit a reduction of the communication overhead.

In designing our negotiation protocols, although based on general principles whose implementation should be easily portable among different platforms, we have taken a pragmatic view, and started from the analysis of the capabilities of popular, off-the-shelf sensor nodes, thus providing a proof-of-concept protocol implementation on actual wireless sensor motes. We have specifically focused on two family of nodes: The *TelosB* and *MICA2* motes. TelosB features an 8MHz MSP430 micro-controller, a 16b RISC processor and the IEEE 802.15.4 compliant transceiver, the Chipcon CC2420. The *MICA2* motes are equipped with the 4MHz Atmel ATmega128L 8b micro-controller and the Chipcon CC1000 low-power wireless transceiver.

Our program code has been written in a combination of nesC, C and assembly language. We have used as much as possible existing public domain implementation of the employed cryptographic algorithms.

4.1 Cipher Suite Negotiation Procedure

Figure 3 depicts the negotiation procedure of our protocol. This is similar to that employed in both TLS and IPsec IKE. The node initiating the security association set up, called the Initiator, sends an InitiatorHello message. This message fits a 28B packet, as this is the default payload size in TinyOS, the OS used by both TelosB abd MICA2 motes. The specific fields included in the packet are shown in Figure 4. They are the following.

- The first three bytes of the message specify the handshake protocol and the version used, the type of message among those available in the protocol (Hello, Certificate, KeyExchange, Finished, etc.), and the message size (in bytes).
- The next two bytes provide the network layer identifier

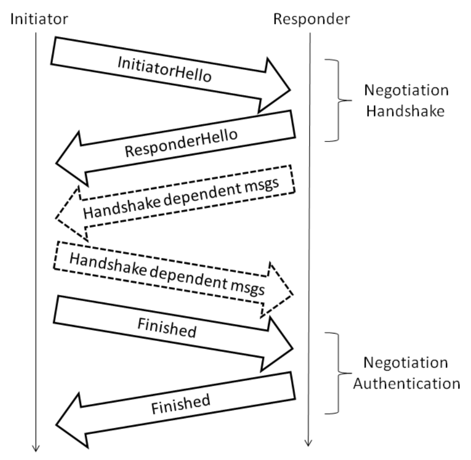

Figure 3: Cipher suite Negotiation procedure.

of the sensor node, enabling multiple protocols at both data link and network layers.

- A session identifier field (two bytes) is used to guarantee that the security association set up among two arbitrary nodes is unique for each party. To this purpose, the two bytes are set by both parties: The Initiator fills the first byte with a value unique for that node, whereas the Responder is in charge to fill the second byte. Note that a security association identifier can use the session identifier as well as the name of the involved peers, i.e., the single byte chosen by a node is not a limiting factor even in the case of nodes, such as a sink, which needs to establish a large amount of security associations.

- For lack of packet payload space, the size of the nonce is reduced to 16B, compared to the 32B of TLS. Instead of using the TinyOS random number generation function (RandomLFSR), nonces have been generated through TinyRNG [19], as this is a cryptographic pseudo-random number generator that uses the received bit errors (unpredictable and difficult to manipulate) as the main source of entropy.

- The final five bytes are reserved for the specification of the supported cipher suites. A compact bit-wise encoding format has been chosen to retain a constant size irrespective of the number of cipher suites signaled. The first two bytes specify the asymmetric key exchange algorithms. Each algorithm is indicated with a bit set to one in the appropriate position $0 \ldots 15$, so that up to 16 asymmetric algorithms can be supported. The next two bytes indicate the symmetric cipher. Again, 15 possible ciphers plus the "null" cipher can be signaled. Finally, the last byte indicates the hash to be used in the message authentication code.

The drawback of the proposed cipher suite specification approach is that a relative preference list for the Initiator, as given in TLS by the order in which cipher suites are listed, cannot be now supported. We therefore rely on an absolute preference list, where stronger ciphers are assumed to be coded as most significant bits in the relevant packet fields.

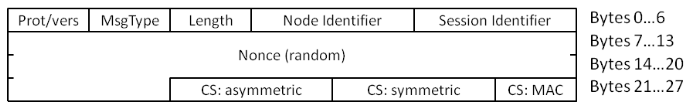

Figure 4: Negotiation packet format.

The ResponderHello message has the same structure, and specifies which (single) cipher suite is employed. As a result, at most three bits in the last five bytes are set to 1.

A difference with respect to TLS is that the closing Finished message is not encrypted using the negotiated symmetric cipher. This is in line with the principle that the proposed key exchange protocol is decoupled from the following data delivery phase, and it permits to save bytes. The Finished message contains an HMAC computed over the whole set of messages exchanged during the handshake, keyed with the master secret computed at the end of the key exchange algorithm, and derived from the Pre-Master Secret exchanged (RSA) or agreed upon (ECDH, IBC) using the usual TLS PRF-based key derivation [4]. The size of the HMAC depends on the digest size of the hash function employed, 16B or 20B for MD5 or SHA-1, respectively, and hence fits the packet payload size (if hash functions with larger digest are eventually employed, truncation to fit the packet size is advised).

4.2 Key Exchange with RSA and ECDH

Messages exchanged after the ResponderHello depend on the negotiated asymmetric algorithm, and may include a Certificate message and/or a KeyExchange message. To cope with the stringent resource limits of sensor motes, the main target of the protocol implementation consisted in reducing as much as possible the number and size of the exchanged messages. In the case of RSA key transport or ECDH key agreement, the number of messages cannot be reduced with respect to the TLS handshake (besides the straightforward removal of the ServerHelloDone TLS-equivalent message, which could be managed by using header bits). Hence, the most effective reduction concerns the size of the messages themselves, and especially of the Certificate message size. For the RSA-1024 case, a certificate of at least 262B is needed [10], whereas an ECC-160 certificate can be packed into as little as 86B [10]. These sizes do not prevent message fragmentation to occur because of the TinyOS 28B payload size. However, they provide a consistent reduction with respect to the relevant X.509v3 certificates (approximately 700B for RSA-1024 and 530B for ECC-160).

In our implementation the cryptographic algorithms considered, namely, RSA public key encryption, ECDH key agreement, and ECDSA signatures, have been adapted from the public domain libraries RELIC [20] and TinyECC [21].

4.3 Key Exchange with IBC

Our negotiation protocol can easily support key exchange based on Identity Based Cryptography (IBC). As mentioned earlier, IBC allows the nodes to exchange significantly less messages as the protocol requires only the exchange of four messages (the two Hello from both Initiator and Responder, and the two Finished messages for negotiation authentication).

Table 1: Energy consumption of cryptographic algorithms [mJ].

Mote	Algorithm	Signature		Key Exchange	
		Sign	Verify	Initiator	Resp.
MICA2	RSA-1024	292.0	11.8	16.7	292.0
	ECC-160	23.4	44.5	25.2	25.2
	ID-NIKDS	/	/	49.2	49.2
TelosB	RSA-1024	68.9	2.7	3.5	68.9
	ECC-160	6.3	12.4	6.2	6.2
	ID-NIKDS	/	/	18.1	18.1

We have exploited the very efficient and compact ID-NIKDS implementation [22] of IBC for MICA2 motes, and our own custom implementation for the TelosB nodes. Computation of the Pre-Master Secret is performed once the node identifier (the public key for IBC) is received at both sides. At the Initiator side, the Pre-Master Secret is computed as $PMS_{I,R} = \hat{e}(S_I, P_R)$, whereas at the Responder side the same secret is computed as $PMS_{R,I} = \hat{e}(S_R, P_I)$. Here, $PMS_{R,I} = PMS_{I,R}$ is a 24B Pre-Master Secret computed by the pair of nodes and $P_X = \phi(id_X)$ is the public key of node X directly derived from its identifier through an hash function $\phi(.)$ that maps the node identifier onto the Elliptic curve[2]. S_X is the private key assigned to node X by a trusted third party deploying the IBC scheme. Finally, $\hat{e}(S_X, P_Y)$ is the pairing function employed. In our case this function is η_T pairing computed over a supersingular Elliptic Curve. These types of curves are particularly appealing for WSN environments, as they permit an extremely efficient implementation in terms of memory and processing requirements, when compared with implementations on ordinary curves [23, 24].

5. EXPERIMENTAL PERFORMANCE ASSESSMENT

To assess the practical feasibility of our proposed security negotiation protocol we have first computed the communication overhead associated to our protocol when RSA, ECC or ID-NIKDS are selected by the two peers. We have also measured the energy consumed during the negotiation phase, distinguishing between the energy needed to compute the cryptographic primitives and the energy due to message exchange.

Table 1 shows the energy consumed for the considered cryptographic algorithms. In order to estimate the energy needed for signature generation and verification we use the formula $E = U * I * t$. For MICA2 motes, when the processor is in active mode $I = 8mA$. TelosB consumes $I = 1.8mA$ when in active mode. Typically, $U = 3.0V$ with two new AA batteries[3].

[2] In our implementation we have assigned node identifiers so that the hash $\phi(.)$ provides a point over the elliptic curve, i.e., so that the resulting quadratic equation needed to derive the y-axis value from an x-coordinate has a solution. Note that, in general, this may not be guaranteed by an arbitrary node identifier.

[3] The energy consumption of the cryptographic operations has also been obtained experimentally by taking the voltage difference between the sensor and a set of resistors used to clean up the signal from the noise coming from the lab environment. During the measurements the sensor was powered by a 3V generator. We have observed a negligible difference between the used formulas and the actual measurements.

Table 2: Energy cost of handshake.

Handshake overhead			
Algorithm	Communication [*byte*]	Energy cost [mJ]	
		MICA2	TelosB
RSA	468	364.88	89.98
ECC	276	151.46	43.68
ID-NIKDS	100	110.62	39.19

During a RSA handshake the Initiator performs two operations: Responder's certificate verification, and encryption of the Pre-Master Secret with the responder public key. The Responder instead executes only the operation of decrypting the KeyExchange message delivered by the Initiator. When using ECC the Initiator verifies the ECDSA signature on the Responder certificate and executes an ECDH operation to obtain the Pre-Master Secret. Identical procedures are executed at the Responder side. With ID-NIKDS, both Initiator and Responder are required to perform a bilinear pairing to compute the Pre-Master Secret.

The total energy consumption during the handshake is listed in Table 2, which also shows the total overhead in bytes. Results are separately displayed for RSA, ECC and ID-NIKDS and refer to our implementation both on the TelosB and on the MICA2 platforms. The protocol with the higher overhead is RSA. Using ID-NIKDS the overhead decreases by a factor of four. As for energy consumption, RSA is also the worst solution, since ECC and ID-NIKDS consume around three times less (see also Figure 9, commented below). The difference of energy consumption between ECC and ID-NIKDS is due to the higher number of operations of ECC and to its higher communication overhead. The improvement of ID-NIKDS over ECC on MICA2 motes is a order of magnitude higher than the same on TelosB motes because of the tailored code optimization of ID-NIKDS realized through RELIC libraries. This optimization is not available for TelosB motes.

Figures 5 and 6 depict the percentage of energy consumed by the Initiator and by the Responder, respectively, during the handshake phase, on the two mote platforms considered. This allows us to quantify how much we pay to support the negotiation of the cryptographic key management approach itself. In all cases, the cost imposed by public key cryptography is by far the dominant one: 88% for RSA, 83% with ECC, and 92% using ID-NIKDS on MICA2 motes. Similar figures are obtained on the TelosB platform: 84% with RSA, and 79% and 93% using ECC and ID-NIKDS, respectively. Conversely, the cost of the negotiation ranges between 2% and 7% of the handshake total cost.

Figures 7 and 8 show the fraction of the energy cost corresponding to the handshake over the global cost of handshake and data communication. As expected, as the amount of communicated data (i.e., the transaction size) increases, the impact of the handshake decreases. On MICA2 motes, the communication cost is higher than that of the handshake after 4KB have been transmitted for RSA, 1.7KB with ECC and 1.2KB with ID-NIKDS. On TelosB nodes, instead, using RSA the handshake cost becomes less than the communica-

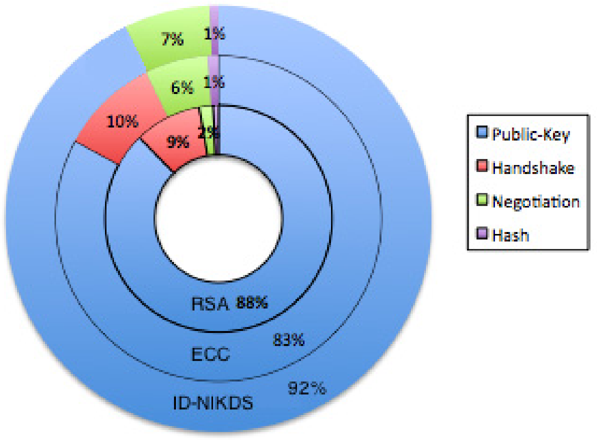

Figure 5: Decomposition of energy consumption on Mica2 motes.

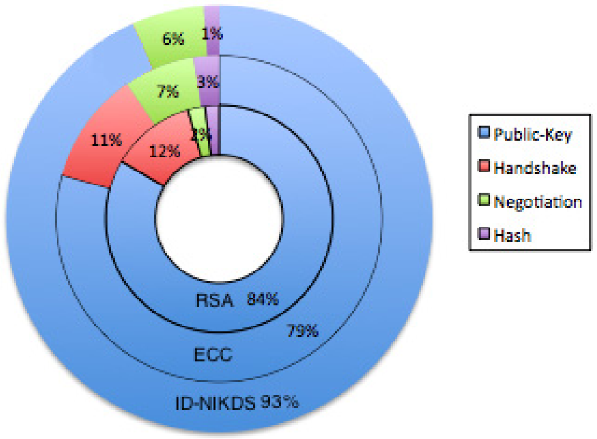

Figure 6: Decomposition of energy consumption on TelosB motes.

tion cost after 3.3KB, the ECC cost after 1.6KB and 1.5KB are enough when using ID-NIKDS.

Figure 9 shows the execution time of the algorithms for public key cryptography used throughout the handshake. RSA is, as well expected, the slower solution on both platforms, although the time it executed on the TelosB nodes is half of the time it takes to run it on the MICA2 motes, because the former have a faster processor than the latter (8MHz vs. 4MHz and 16b vs. 8b). On MICA2 motes executing ECC and ID-NIKDS take very similar times: 2.2s and 2.05s, respectively. On the TelosB motes the difference is greater: 1.7s for ECC and 3.33s for ID-NIKDS. Less time is required to run ID-NIKDS on the MICA2 nodes because its implementation is based on the RELIC libraries, which are more efficiently implemented than the MIRACL libraries used on the TelosB.

Finally, Table 3 shows the memory occupancy of the code for the considered protocols, and for both MICA2 and TelosB motes (supporting 128KB and 48KB ROM, respectively). The asymmetric cryptographic protocols are the most space

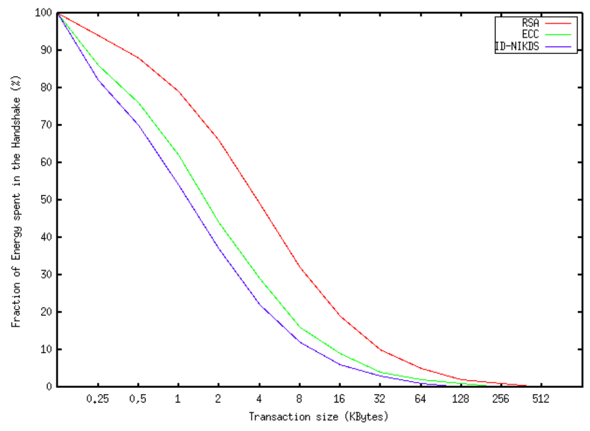

Figure 7: Fraction of the Energy Spent in the Handshake Depending on the Transaction Size on Mica2 motes.

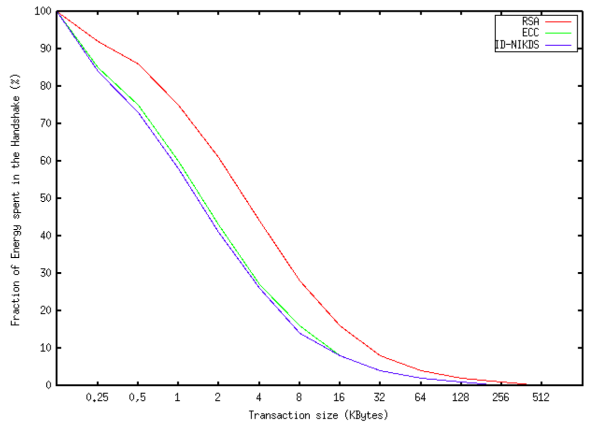

Figure 8: Fraction of the Energy Spent in the Handshake Depending on the Transaction Size on TelosB motes.

consuming, total occupancy being about four times higher than what needed by the rest of our protocol (included the hash function, SHA1 on Table 3).

6. CONCLUSIONS AND FUTURE WORK

We have investigated efficient and flexible security negotiation protocols for WSNs. Based on typical characteristics of WSNs and of sensor nodes we have proposed to adopt a tailored TLS handshake for these networks. A security association set up protocol has been detailed and its implementation and feasibility has been assessed through experiments on MICA2 and TelosB nodes. Our results, carried on the two platforms by using different public key cryptography protocols (RSA, ECC and ID-NIKDS), have shown that the energy consumption of the proposed negotiation phase is minimal with respect to the cost needed for the execution of the protocol itself. Therefore, the advantages that the negotiation introduces, maximum flexibility in primis, are worth the minimal extra cost.

An important research and deployment direction opened by our work consists in adapting leading WSN security pro-

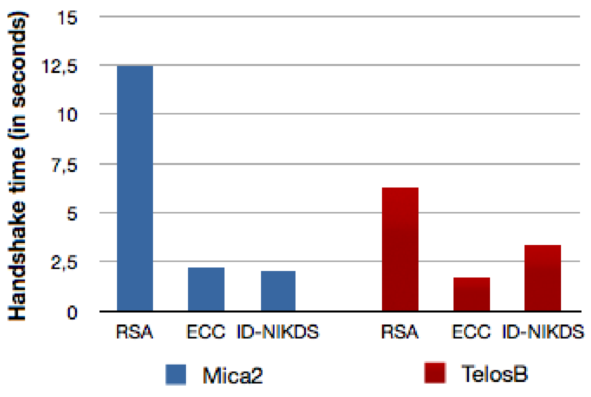

Figure 9: Handshake time on *MICA2* and *TelosB* motes.

Table 3: Code size.

	Code size [bytes]	
	MICA2	TelosB
RSA	12,542	7,350
ECC	29,976	18,148
ID-NIKDS	20,344	16,744
SHA1	3,834	2,602
Our protocol	13,426	10,788

tocols such as TinySec and MiniSec, so that their operation is governed by a dynamically established session state, i.e., taking advantage of a preliminary security association set up phase comprising cipher suites negotiation provided by our proposed protocol.

7. ACKNOWLEDGMENTS

This work is partially supported by the ARTEMIS project #1000128 CHIRON (Cyclic and person-centric Health management: Integrated appRoach for hOme, mobile and clinical eNvironments).

8. REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, “Wireless sensor networks: a survey”, Computer Networks, Vol. 38(4), Mar. 2002, pp. 393-422.

[2] C. Garcia-Hernandez, P. Ibarguengoytia-Gonzales, J. Garcia-Hernandez, J. Perez-Diaz, “Wireless Sensor Networks and Applications - a Survey”, Int. J. of Computer Science and Network Security, Vol. 7(3), Mar. 2007, pp. 264-273.

[3] C. Kaufman, editor, “Internet Key Exchange (IKEv2) Protocol”, IETF RFC 4306, Dec. 2005.

[4] T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol, Version 1.2”, IETF RFC 5246, Aug. 2008.

[5] A. Shamir, “Identity-based cryptosystems and signature schemes”, Proc. of CRYPTO 84 on Advances in cryptology, Santa Barbara, CA, USA, 1985, pp. 47-53.

[6] D. Malan, M. Welsh, M. Smith, “A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography”, IEEE Sensor and Ad Hoc Communications and Networks, SECON 2004, pp. 71-80.

[7] G. Gaubatz, J. P. Kaps, B. Sunar, “Public key cryptography in sensor networks”, 1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004).

[8] C. Karlof, N. Sastry, D. Wagner, “TinySec: a link layer security architecture for wireless sensor networks”, Proc. of the 2nd Int. Conf. on Embedded networked sensor systems, SenSys 2004, Baltimore, MD, USA, pp. 162-175.

[9] M. Luk, G. Mezzour, A. Perrig, V. Gligor, “MiniSec: a secure sensor network communication architecture”, 6th Int. Conf. on Information processing in sensor networks, IPSN 2007, Cambridge, MA, USA, 2007, pp. 479-488.

[10] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, S. C. Shantz, “Sizzle: a standards-based end-to-end security architecture for the embedded Internet”, Sun Microsystems, Inc., Technical Reports, SERIES 13103, 2005.

[11] S. Fouladgar, B. Mainaud, K. Masmoudi, H. Afifi, “Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks”, Springer LNCS, Vol. 4357/2006, Security and Privacy in Ad-Hoc and Sensor Networks, Mar. 2007, pp. 32-42.

[12] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, B. Moeller, “Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)”, IETF RFC 4492, May 2006.

[13] E. Rescorla, N. Modadugu, “Datagram Transport Layer Security”, IETF RFC 4347, April 2006.

[14] R. Rivest, A Shamir, L. Adleman, “A method for obtaining Digital Signatures and Public Key Cryptosystems”, Communications of the ACM. Feb., 1978 21(2) pages 120-126.

[15] A. K. Lenstra, E. R. Verheul, “Selecting Cryptographic Key Sizes”, Journal of Cryptology: the journal of the International Association for Cryptologic Research, 2001.

[16] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, “Recommendation for Key Management - Part 1: General (Revised)”, NIST Special Publication 800-57, March 2007.

[17] S. D. Galbraith, K. G. Paterson, N. P. Smart, “Pairings for cryptographers”, Discrete Applied Mathematics, Vol. 156(16), 2008, 3113-3121.

[18] B. Parno, A. Perrig, V. Gligor, “Distributed Detection of Node Replication Attacks in Sensor Networks”, Proc. of the IEEE Symp. on Security and Privacy, Oakland, CA, May, 2005.

[19] http://www.ist-ubisecsens.org/downloads/tinyrng/tinyrng.php

[20] http://code.google.com/p/relic-toolkit/

[21] http://discovery.csc.ncsu.edu/software/TinyECC/

[22] R. Sakai, K. Ohgishi, M. Kasahara, “Cryptosystems based on pairing”, Symp. Cryptography and Information Security, SCIS 2000, Jan 2000, pp. 26-28.

[23] P. Barreto, S. Galbraith, C. Heigeartaigh, M. Scott, “Efficient pairing computation on supersingular abelian varieties”, Designes Codes And Cryptography, 2006.

[24] L. B. Oliveira, M. Scott, J. Lopez, R. Dahab, “TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks”, Networked Sensing Systems, 2008, pp. 173-180.