The Energy Cost of Cryptographic Key Establishment in Wireless Sensor Networks[*]

Johann Großschädl
jgrosz@iaik.tugraz.at

Alexander Szekely
aszekely@iaik.tugraz.at

Stefan Tillich
stillich@iaik.tugraz.at

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria

ABSTRACT

Wireless sensor nodes generally face serious limitations in terms of computational power, energy supply, and network bandwidth. Therefore, the implementation of effective and secure techniques for setting up a shared secret key between sensor nodes is a challenging task. In this paper we analyze and compare the energy cost of two different protocols for authenticated key establishment. The first protocol employs a "light-weight" variant of the Kerberos key distribution scheme with 128-bit AES encryption. The second protocol is based on ECMQV, an authenticated version of the elliptic curve Diffie-Hellman key exchange, and uses a 256-bit prime field GF(p) as underlying algebraic structure. We evaluate the energy cost of both protocols on a Rockwell WINS node equipped with a 133 MHz StrongARM processor and a 100 kbit/s radio module. The evaluation considers both the processor's energy consumption for calculating cryptographic primitives and the energy cost of radio communication for different transmit power levels. Our simulation results show that the ECMQV key exchange consumes up to twice as much energy as the Kerberos key distribution. However, in large-scale networks, ECMQV is more energy-efficient than Kerberos.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General —*Security and Protection*; E.3 [**Data**]: Data Encryption.

General Terms

Algorithms, Design, Performance, Security.

Keywords

Wireless networking, security protocols, cryptography, key establishment, energy evaluation.

1. INTRODUCTION

Wireless sensor networks are, in general, more vulnerable to attacks and unauthorized access than traditional (wired) networks. The security threats stem primarily from two key characteristics of wireless sensor networks. First, the sensor nodes are often deployed in hostile environments where they can be easily captured, compromised, or manipulated by an adversary. Second, the use of radio waves as data transmission medium enables an adversary to eavesdrop on the communication, inject false messages, and replay or change old messages. This multitude of threats calls for effective protection mechanisms. However, the "traditional" security architectures used in wired networks are not directly applicable to wireless sensor networks because of the resource constraints of sensor nodes, most notably limited energy supply, computational power, and network bandwidth.

In the recent past, a multitude of security architectures and protocols have been proposed to increase the resistance against the aforementioned attacks, taking the special characteristics of wireless sensor networks into account [19]. Key management and authentication are the two most essential security services to maintain the proper operation of a sensor network. The establishment of a shared secret key between a pair of nodes is the basis for other security services such as encryption. A simple approach for key establishment is to load one or more secret keys onto each sensor node prior to deployment. Most practical security protocols based on pre-deployed keying use either a single network-wide key shared by all nodes [10] or a set of keys randomly chosen from a key pool such that two nodes will share (at least) one key with a certain probability [7]. These protocols are easy to implement and entail only little overhead since no complex key agreement has to be performed. However, most protocols based on pre-deployed keys have disadvantages regarding scalability and vulnerability to node capture.

1.1 Key Establishment Protocols

Key establishment in sensor networks can also be realized with protocols where the nodes set up a shared secret key after deployment, either through *key transport* or *key agreement* [13]. A key transport protocol is a protocol where one entity creates or otherwise obtains a secret key and transfers it securely to the other entity (or entities). Key agreement refers to a mechanism or protocol where all participating entities contribute a random input which is used to derive a shared secret key. The advantage of key agreement over key transport is that no entity can predetermine the resulting key as it depends on the input of all participants.

In practice, many key establishment protocols involve a so-called *trusted third party* or *trusted authority* to set up a shared key between two entities. Examples of such protocols include the Needham-Schroeder protocol [15], the Kerberos key distribution protocol [11], and the SPINS node-to-node key establishment protocol [16]. All these protocols build on

[*]The research described in this paper has been supported by the Austrian Science Fund (FWF) under grant number P16952-NO4 and by the European Commission under grant number FP6-IST-033563 (project SMEPP).

secret-key cryptography (symmetric encryption) and need a trusted party (in the following abbreviated as T) with which each entity shares a long-term secret key *a priori*, i.e. the long-term key is pre-distributed over a secure channel. The entities use T to prove their identity (authentication) or to generate and transmit a *session key* that allows two entities to securely communicate with one another. Depending on the protocol, T either provides the session key by itself or makes a session key generated by one entity available to the other by encrypting it with the long-term key shared with the latter.

Key agreement (or key exchange) protocols do not rely on a third party to set up a shared secret key. Instead, they allow two entities to directly establish a key by exchanging messages over an insecure communication channel. Another difference between T-based key establishment (Kerberos) and key agreement is that the latter does not require a pre-distribution of keying material, and hence it also works for entities which have never met in advance or do not possess a secret key with a trusted third party. Most practical protocols for key agreement use asymmetric techniques (i.e. public-key cryptosystems). A well-known example of a protocol providing non-authenticated key agreement is the Diffie-Hellman protocol [6], which can be designed around a multiplicative group of integers modulo a large prime or an additive group of rational points on an *elliptic curve* defined over a finite field [2, 8]. The security of the Diffie-Hellman cryptosystem relies on the presumed hardness of the discrete logarithm problem (DLP) in a group of large order.

1.2 Contributions of this Paper

While the recent literature contains a number of papers about key establishment protocols for wireless sensor and ad-hoc networks (see [19] and the references therein), only very few of these actually analyze and compare the pros and cons of different protocols. Especially the energy-efficiency of key establishment protocols has not been widely explored until now. The only relevant publications we are aware of are [4], [9], [17], and [24]; the former two compare the overall energy cost of Kerberos key establishment and Diffie-Hellman key agreement. It was concluded in [9] that key agreement protocols using public-key cryptography (e.g. elliptic curve systems) require between one and two orders of magnitude more energy than a Kerberos-like protocol.

The high energy requirements of public-key algorithms reported in [4, 9] have raised serious concerns about their feasibility in wireless sensor networks. However, it must be considered that the experiments documented in [4] and [9] were conducted in 2000 and 2002, respectively, and since then, enormous progress has been made in the efficient implementation of public-key cryptography, especially elliptic curve cryptography [8]. This progress makes it necessary to completely re-evaluate the energy requirements of elliptic curve cryptosystems. In the present paper we demonstrate that the energy cost of elliptic curve cryptography—when implemented with state-of-the-art algorithms—is far lower than reported in previous work. Our experimental results clearly show that key exchange protocols using elliptic curve systems are feasible for wireless sensor networks. Even more important, we found that, in large sensor networks, elliptic curve-based key exchange may actually require *less* energy than Kerberos key distribution.

1.3 Methodology and Tools

We conducted our evaluation of the energy cost of cryptographic key establishment on a WINS sensor node from Rockwell Scientific [1]. The WINS node is equipped with a 133 MHz StrongARM processor and features a 100 kbit/s radio module. Our motivation for using this specific sensor node is twofold. First, we wanted to use the same node as the authors of [4] and [9] so that we can directly compare the results and, in this way, demonstrate the progress in public-key cryptography in recent years. The second reason for choosing the WINS node is because ARM processors have a considerable market share in the embedded systems field. Besides the WINS node, ARM processors are used in a number of other sensor nodes like the Intel iMote or the Sun SPOT. In addition, various PDAs and mobile phones have (Strong)ARM processors. Therefore, we hope that our study of the energy cost of key establishment protocols will be useful not only for the sensor network community, but also for researches in other fields like mobile computing.

Our evaluation of key establishment protocols considers both the energy that the StrongARM consumes during the execution of cryptographic algorithms and the energy cost of radio communication. We used the energy characteristics of the WINS node reported in [20], upon which the node's average power consumption is 360 mW when the processor is active. The energy required for the calculation of cryptographic primitives is simply the product of the average power consumption and the execution time. We determined the execution time of the cryptographic primitives through simulations with SimIt-ARM, a cycle-accurate instruction set simulator for the StrongARM [18]. The main advantage of a simulation-based energy evaluation is that the results can be easily re-produced, even by researchers who do not have access to a "real" WINS node.

2. THE ENERGY COST OF CRYPTOGRAPHIC KEY ESTABLISHMENT

The overall energy cost of a key establishment protocol is not only determined by energy required for calculating cryptographic primitives, but also by the energy cost of radio communication between the involved parties. Raghunathan et al [20] analyzed the power consumption characteristics of modern sensor nodes and found out that, depending on the operating modes of the node's components, the wireless transmission of data can account for a major portion of the total energy consumption. For example, the WINS sensor node [1] from Rockwell Scientific consumes between 1500 and 2700 times more energy for the transmission of one bit over the RF module than for the execution of an instruction on the node's processor unit (the exact factor depends on the transmit power level).

Kerberos and similar key establishment protocols using a trusted third party have the advantage that they can be implemented solely with secret-key primitives, and hence do not require to perform computation-intensive (and battery-draining) cryptographic computations. The penalty of these protocols is the three-way communication since two entities wishing to set up a secret key do not only transmit messages to each other but also to the trusted authority. Thus, the communication energy cost of Kerberos-like protocols is typically much higher than the energy required for calculating cryptographic primitives [4]. On the other hand, the relation between communication energy and computation energy is completely different for key agreement protocols using public-key cryptography. The Diffie-Hellman protocol requires to perform computation-intensive (and hence energy-costly) arithmetic operations like exponentiation in a multiplicative group or scalar multiplication in an additive group, whereby the group order is at least 1024 bits for the former and 160 bits for the latter, respectively. Regarding communication energy cost, the Diffie-Hellman protocol has the advantage that the two entities participating in the key establishment process communicate directly with each other, i.e. there is no third party involved. In summary, the Diffie-Hellman key exchange is characterized by high energy consumption for calculating cryptographic primitives, but relatively low communication energy cost [9].

2.1 Simulation Results from Previous Work

Carman et al [4] conducted an extensive study of cryptographic primitives and security protocols for wireless sensor networks, including an energy analysis of various public-key cryptosystems like RSA and DSA. They found out that the energy cost for generating a 1024-bit RSA signature is about 15 mJ on a Rockwell WINS sensor node [1] equipped with a 133 MHz StrongARM processor. This result was achieved by applying the Chinese Remainder Theorem (CRT) for the modular exponentiation[1]. Consequently, it can be estimated that the energy cost of a 1024-bit modular exponentiation without CRT, such as required for the Diffie-Hellman key exchange, is approximately 60 mJ on the StrongARM. The energy consumption for transmitting a 1024-bit data block using the WINS radio subsystem operating at 10 kbit/s and 10 mW transmit power is roughly 21.5 mJ on the sending node and 14.3 mJ on the receiving node, respectively [4]. In other words, the computation energy cost is nearly twice the communication energy cost. For comparison, encrypting a 128-bit block of data with a symmetric algorithm like the Advanced Encryption Standard (AES) needs only 2.17 μJ on the StrongARM, which is virtually negligible in relation to the afore-mentioned energy values.

Hodjat and Verbauwhede [9] analyzed and compared the energy cost of elliptic curve Diffie-Hellman (ECDH) key-exchange and Kerberos-like key establishment on a Rockwell WINS node. The energy consumption of the ECDH protocol was evaluated for three different key lengths: 128, 192, and 256 bits. Accordingly, Hodjat and Verbauwhede used the binary extensions fields GF(2^{128}), GF(2^{192}), and GF(2^{256}) as underlying algebraic structure for the elliptic curves. On the WINS node, the average energy cost for computing a point multiplication[2] is 300 mJ, 1070 mJ, and 2340 mJ for the 128-bit, 192-bit, and 256-bit field, respectively. When considering both the computation and the communication energy cost, the ECDH key exchange requires 1213.7 mJ, 4296.0 mJ, and 9378.3 mJ for a key length of 128, 192, and 256 bits, respectively. Hodjat and Verbauwhede also analyzed the energy consumption of a simplified variant of the Kerberos key distribution using AES encryption. The overall energy cost was estimated to be less than 140 mJ (for 128, 192, and 256-bit keys), whereby the energy needed for AES encryption/decryption is almost negligible in relation to the communication energy cost.

Since the security of a 256-bit elliptic curve cryptosystem corresponds to the security of 128-bit AES encryption, it can be concluded that, for comparable security levels, the ECDH key agreement requires about 67 times more energy than the Kerberos-like key transport protocol.

3. LIGHTWEIGHT KERBEROS PROTOCOL WITH SHORT MESSAGES

Kerberos is a network authentication system that uses a *trusted third party* (or *trusted authority*) to authenticate two entities (i.e. to prove their identity to one another) by issuing a shared *session key* between them [11]. This idea is originally due to Needham and Schroeder and was first published in [15]. Roughly speaking, Kerberos extends the Needham-Schroeder protocol by using timestamps and lifetimes to indicate the freshness of a message, which helps to prevent replay attacks. The primary purpose of Kerberos is to authenticate clients and servers in a network; the establishment of a shared key is a side effect. Kerberos can be implemented with secret-key cryptography and is relatively easy to manage due to its centralized design. On the other hand, Kerberos is less scalable than key agreement protocols using public-key cryptography.

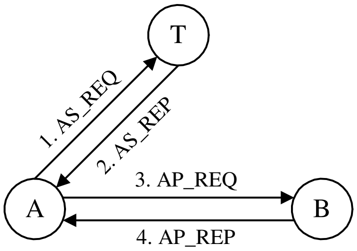
Each entity on the network shares a long-term secret key with the trusted third party (in the following abbreviated by T), which enables the entities to verify that messages from T are authentic. Similarly, knowledge of the long-term key also serves to prove an entity's identity to T. To set up a shared key between two entities A and B, the trusted third party T generates a session key and securely delivers it to A and B encrypted in the long-term key that T shares with A and with B, respectively. Extraction of the session key is only possible for the legitimate entities which possess the corresponding long-term key. Thus, by trusting T, entities can authenticate each other (i.e. prove that they are who they claim to be) and establish a session key.

In a sensor network, the long-term key that each node shares with T is pre-deployed, but contrary to approaches with a single network-wide key, each node has a unique key and thus the capture of some sensors by an adversary does not jeopardize the security of other sensors. The trusted third party can be a single centralized entity (such as the base station or a cluster head) or distributed among trusted sensor nodes [4].

Kerberos, as specified in [11], is a full-featured protocol originally designed for mutual client/server authentication in "wired" networks. The messages exchanged in Kerberos can have a payload of several kilobytes, which makes the standard Kerberos protocol rather unpractical for use in sensor networks where data transfer is extremely costly in terms of energy consumption. Therefore, we introduce in this section a lightweight version of the original Kerberos protocol optimized for small messages. In a nutshell, our lightweight key establishment protocol can be described as "simplified Kerberos without ticket granting service." Figure 1 illustrates the message transfers between entity A and B and the trusted third party T, assuming that A wishes to establish a session key with entity B. Both A and B share a long-term secret key with T, similar to the conventional Kerberos protocol.

[1] Using the CRT allows to split a 1024-bit modular exponentiation into two 512-bit exponentiations, which reduces both the execution time and energy by a factor of four.

[2] A point multiplication is the fundamental operation of all elliptic curve cryptosystems, comparable to modular exponentiation in "traditional" cryptosystems like RSA.



1. AS_REQ: A, B, n_A

2. AS_REP: $\{ k_{AB}, B, t_S, t_E, n_A \} k_{AT}$, $\{ k_{AB}, A, t_S, t_E \} k_{BT}$

3. AP_REQ: $\{ k_{AB}, A, t_S, t_E \} k_{BT}$, $\{ A, t_A \} k_{AB}$

4. AP_REP: $\{ t_A \} k_{AB}$

Figure 1: Simplified Kerberos protocol exchange (an expression of the form $\{X\}\,k$ means that message X is encrypted using the key k.

In the first message (AS_REQ) entity A asks the trusted third party T for a session key that enables A to securely communicate with B. Thereupon, T generates a session key k_{AB} and assembles a reply message (AS_REP) consisting of a *ticket* and other data as illustrated in Figure 1. The ticket contains the freshly generated session key k_{AB} and is encrypted in the long-term key k_{BT} shared between B and T. Besides the ticket, the AS_REP message also includes the session key k_{AB} in a form readable by A, i.e. encrypted in the long-term key k_{AT} shared between A and T. Having received AS_REP, entity A decrypts the non-ticket portion of the message to obtain the session key k_{AB}. Next, A produces an *authenticator*, encrypts it using the key k_{AB}, and sends it along with the ticket to entity B (message AP_REQ in Figure 1). Entity B first decrypts the ticket using its long-term key k_{BT} and extracts the session key k_{AB}. The session key enables B to decrypt the authenticator from the AP_REQ message, which, if successful, proves A's identity since only a legitimate entity possessing k_{AB} can generate a valid authenticator. Finally, message AP_REP, sent from B to A, confirms B's true identity and completes the mutual authentication.

Besides session key, ticket, and authenticator, the messages sent in our lightweight Kerberos protocol also include other data like node IDs, nonces, timestamps, or expiration times in order to guarantee a message's freshness and to prevent replay attacks. However, all these fields increase the length of a message and, consequently, the energy required for its transmission. Appendix A contains a more detailed description of the messages and the rationale behind their content. Though our lightweight Kerberos is similar to the standard Kerberos protocol, we feel that it is necessary to describe all messages in detail to help the reader understand the evaluation of the communication energy cost which we conduct in Section 5.

4. AUTHENTICATED DIFFIE-HELLMAN KEY EXCHANGE

The primary advantage of Kerberos-like protocols is that they can be implemented with secret-key cryptosystems and hence do not require to perform computation-intensive and energy-costly public-key algorithms. However, the Kerberos protocol relies on the sensor nodes' ability to communicate with the trusted third party (base station) during the key establishment process. On the other hand, Diffie-Hellman key exchange [6] does not involve a third party to establish a secret key between two entities, which reduces the communication overhead. The Diffie-Hellman protocol allows two entities to jointly generate a shared secret key by exchanging messages directly with each other.

The Diffie-Hellman key exchange can be embedded into any cyclic group in which the discrete logarithm problem (DLP) is hard. Well-known examples of such groups are the multiplicative group $\mathbb{Z}_p^*$ and the additive group of points defined by an elliptic curve over a finite field [8]. The elliptic curve Diffie-Hellman (ECDH) has a number of advantages over the standard Diffie-Hellman in $\mathbb{Z}_p^*$, including a "higher security per bit" (i.e. shorter keys) and a multitude of implementation options. Appendix B summarizes the basics of Diffie-Hellman as well as ECDH key exchange and gives a brief introduction to *elliptic curve cryptography.* A more detailed discussion of these topics can be found in [2, 8].

The Diffie-Hellman key exchange in its original form, as described in Appendix B, does not authenticate the entities A and B, and is therefore vulnerable to man-in-the-middle attacks [13]. A straightforward way to authenticate A and B to each other is to sign the two messages in the Diffie-Hellman key exchange. However, the problem with signed Diffie-Hellman is that it is wasteful in bandwidth since a signature needs to be attached to the messages transferred between A and B. An alternative approach is to keep the message flow as in the original Diffie-Hellman protocol, but change the way in which the shared key is derived with the goal to provide *implicit authentication.* Typical examples of key exchange protocols with implicit authentication are the Menezes-Qu-Vanstone (MQV) protocol, and its elliptic curve variant, the ECMQV protocol [12]. Both are extensions of the standard Diffie-Hellman protocol.

In the ECMQV protocol each entity is assumed to have both a *static* (i.e. long-term) public/private key pair and an *ephemeral* (i.e. short-term) key pair. The static key pair is valid for a certain period of time, whereas new ephemeral keys are generated for each run of the protocol. A shared secret is derived using the static keys and the ephemeral keys, which guarantees that each protocol run between two entities A and B produces a different shared secret. In what follows, we assume that every entity of the network knows the elliptic curve domain parameters p, α, β, n, and G (an explanation of the parameters is given in Appendix B). The static key of entity A consists of a random number a and the point $S = a \cdot G$, whereby the former is the secret part and the latter the public part of the key pair (a, S). Entity B has the static key pair (b, T) consisting of the secret key b and the public key $T = b \cdot G$. The entities participating in the ECMQV protocol first exchange the public part of their static keys, i.e. entity A sends S to B, and entity B sends T to A. However, the static keys are long-term keys which are generated only once, for example during the initialization phase of the network. Therefore, the generation and transmission of the public parts S and T does not fall into account in the actual ECMQV key establishment process described below.

After having exchanged the public parts of their static keys, entity A and B perform the following steps to agree on a shared secret [3]. First, entity A generates the ephemeral key pair (c, U), whereby $U = c \cdot G$, and entity B generates the ephemeral key pair (d, V) with $V = d \cdot G$. They exchange the public parts of these ephemeral keys as in the standard ECDH protocol, i.e. entity A sends U to B, and entity B sends V to A. Hence, the messages sent between A and B are exactly the same as in the ECDH protocol. After this exchange, entity A knows its own secret keys a, c, and the public keys S, T, U, and V. On the other hand, entity B knows b, d, S, T, U, and V. The shared secret K is then determined by entity A as specified in Algorithm 1 [3].

Algorithm 1: ECMQV key derivation for entity A.

```
Input: Elliptic curve domain parameters $p$, $\alpha$, $\beta$, $n$, $G$, the
    secret keys $a$, $c$, and the public keys $S$, $T$, $U$, $V$.
Output: A secret point $K \in \mathbb{E}$ shared with the entity with
    public static key $T$.
$m \leftarrow \lceil \log_2(n) \rceil / 2$   {$m$ is the half bitlength of $n$}
$u_A \leftarrow (u_x \bmod 2^m) + 2^m$   {$u_x$ is the $x$-coordinate of $U$}
$s_A \leftarrow (c + u_A a) \bmod n$   {implicit signature}
$v_A \leftarrow (v_x \bmod 2^m) + 2^m$   {$v_x$ is the $x$-coordinate of $V$}
$z_A \leftarrow s_A v_A \bmod n$
$K \leftarrow s_A \cdot V + z_A \cdot T$
```

Entity B can also compute the same value of K by simply swapping the occurrence of (a, c, T, U, V) in Algorithm 1 with (b, d, S, V, U). Thus, entity B obtains

$$u_B = (v_x \bmod 2^m) + 2^m \quad \text{and} \quad v_B = (u_x \bmod 2^m) + 2^m$$

whereby v_x and u_x denote the x-coordinate of the elliptic curve points V and U, respectively. It can be easily observed that $u_B = v_A$ and $v_B = u_A$. Since $s_B = (d + u_B b) \bmod n$ and $z_B = s_B v_B \bmod n$, entity B gets the following value for the shared secret K.

$$\begin{aligned} K &= s_B \cdot U + z_B \cdot S = s_B \cdot (U + v_B \cdot S) = s_B \cdot (U + u_A \cdot S) \\ &= s_B \cdot (c \cdot G + u_A a \cdot G) = s_B \cdot (c + u_A a) \cdot G = s_B s_A \cdot G \end{aligned}$$

Similarly, it can be shown that the secret K computed by entity A (i.e. the quantity $s_A \cdot V + z_A \cdot T$) is nothing else than $s_A s_B \cdot G$, which proves that A and B possess the same secret value K.

The quantity $s_A = (c + u_A a) \bmod n$ used in Algorithm 1 can be viewed as an *implicit signature* for A's ephemeral public key U. It is a "signature" in the sense that the private keys a and c are necessary to compute s_A, and thus only entity A is able to generate this signature. The signature is "implicit" in the sense that entity B indirectly verifies its validity by using $s_A \cdot G = (U + v_B \cdot S)$ to derive the shared secret K (see above equations). Similarly, s_B serves as an implicit signature for B's ephemeral public key V. These implicit signatures allow to achieve mutual authentication in the ECMQV key exchange [8].

Computational Cost of ECMQV Key Exchange

Entity A and entity B generate a new ephemeral key pair for each run of the ECMQV protocol, which requires them to perform a scalar multiplications $k \cdot P$ in order to obtain the ephemeral public keys. The simplest algorithm for computing $k \cdot P$ is the left-to-right binary method (Algorithm 3.27 in [8]), which is also called double-and-add method because it produces the result through a sequence of point doublings and additions, respectively. Given a scalar k consisting of $m = \lceil \log_2(k) \rceil$ bits, the double-and-add method performs exactly m point doublings, whereas the number of point additions depends on the Hamming weight of k. When the scalar k is converted into the *non-adjacent form* (NAF), the computation of $k \cdot P$ requires m point doublings, and, on average, $m/3$ point additions (see [8] for details).

Algorithm 2: Multiple point multiplication.

```
Input: The points $P, Q \in \mathbb{E}$, scalar $k = (k_{m-1}, \ldots, k_1, k_0)_2$
    and scalar $l = (l_{m-1}, \ldots, l_1, l_0)_2$.
Output: $R = k \cdot P + l \cdot Q$.
$Z \leftarrow P + Q$
$R \leftarrow \mathcal{O}$
for $i$ from $m-1$ down to $0$ do
    $R \leftarrow R + R$   {point doubling}
    if $(k_i = 1)$ and $(l_i = 0)$ then $R \leftarrow R + P$ end if
    if $(k_i = 0)$ and $(l_i = 1)$ then $R \leftarrow R + Q$ end if
    if $(k_i = 1)$ and $(l_i = 1)$ then $R \leftarrow R + Z$ end if
end for
return $R$
```

In order to derive the shared secret K, entity A and entity B have to accomplish an operation of the form $k \cdot P + l \cdot Q$ (see Algorithm 1). This operation, which is called multiple scalar multiplication or multiple point multiplication, has an impact on the overall computational cost of the ECMQV key exchange. The naive approach to obtain $k \cdot P + l \cdot Q$ is to compute the point multiplications $k \cdot P$ and $l \cdot Q$ separately, followed by an addition of the results. However, the multiple scalar multiplication $k \cdot P + l \cdot Q$ can be performed much faster when the doublings are combined as shown in Algorithm 2. The idea of this "simultaneous" multiple point multiplication is originally due to E. Straus [23], but often referred to as *Shamir's trick* in the literature.

Algorithm 2 performs m point doublings when m is the bitlength of the scalars k and l. The number of additions depends on the joint Hamming weight of k and l. However, when the scalars k and l are represented in the *joint sparse form* (JSF) [22], the multiple point multiplication can be accomplished with $m/2$ additions on an average. Thus, the computation of $k \cdot P + l \cdot Q$ requires about 17% more point additions than an ordinary scalar multiplication $k \cdot P$, while the number of point doublings is identical.

In summary, the ECMQV key exchange is only slightly more computation-intensive than ECDH. Every run of the ECMQV protocol requires exactly the same bandwidth as the unauthenticated ECDH key exchange, provided that the public parts of the static keys have already been transferred to the other entity.

5. EXPERIMENTAL RESULTS

Sensor nodes are battery driven and hence operate on a strictly limited power budget. The results from previous work [4, 9] show that the overall energy consumption of key establishment protocols depends on the energy required for computing cryptographic primitives and the energy needed for the transmission of messages between sensor nodes. In this section we analyze and compare the energy demands of Kerberos-like key distribution and ECDH/ECMQV key exchange, taking into account both the computation and communication energy cost. Our evaluation is based on the power and energy characteristics of the WINS sensor node developed by Rockwell Scientific [1].

Processor	Sensor	Radio mode	Power
Active	On	Tx, Power: 36.3 mW	1080.5 mW
		Tx, Power: 10.0 mW	910.9 mW
		Tx, Power: 0.96 mW	787.5 mW
		Tx, Power: 0.12 mW	771.1 mW
Active	On	Rx	751.6 mW
Active	On	Idle	727.5 mW
Active	On	Sleep	416.3 mW
Active	On	Removed	383.3 mW
Idle	On	Removed	64.0 mW
Active	Rmvd.	Removed	360.0 mW

Table 1: Power characteristics of the WINS node.

The WINS node has a modular design and consists of a power supply module, a processor module equipped with a StrongARM SA-1100 processor clocked at 133 MHz, a 100 kbit/s radio module capable of operating at different transmit power levels ranging from 1 mW to 100 mW, and a sensor module with an acoustic or seismic sensor. Table 1 is taken from [20] and shows that the overall power consumption of the WINS node depends heavily on the radio module's and the processor's mode of operation. The sensor module has a constant power consumption of 23 mW when it is active.

5.1 Evaluation of Computation Energy

In light of the energy constraints of sensor nodes, it is very important to consider the computational energy cost of the cryptographic primitives. The amount of energy consumed by a primitive on a given processor can be determined as the product of the processor's average power consumption and the time required to execute the primitive. Thus, in order to evaluate the computation energy cost of a cryptographic primitive, we have to determine its exact execution time (e.g. through measurements or with help of a cycle-accurate instruction set simulator like SimIt-ARM [18]) and then calculate the energy as mentioned before.

The amount of power drawn by the StrongARM SA-1100 processor depends on its mode of operation (active, idle, or sleep). When clocked with a frequency of 133 MHz, the StrongARM consumes, on average, about 360 mW in active mode, 40 mW in idle mode, and less than 1 mW in sleep mode. It was reported in [21] that the StrongARM's power consumption is largely dominated by a common overhead (caches, buses, clock tree, decode logic, etc.) and can, as a first order approximation, be regarded as uniform and independent of the executed instructions. Therefore, we simply use the value of 360 mW to evaluate the energy consumed by a cryptographic primitive.

5.2 Evaluation of Communication Energy

In addition to consuming energy through computational processing, security protocols also consume energy due to the transmission of messages. The communication energy depends on the distance between sending and receiving node (which determines the transmit power level) and the time required for sending the message, whereby the latter is proportional to the message length and indirectly proportional to the transmission rate. Furthermore, it must be considered that the transmission of messages consumes energy not only on the sending node, but also on the receiving node.

The radio module of the WINS sensor node can operate in four distinct modes: transmit, receive, idle, and sleep. In general, the radio module consumes significantly less power in sleep mode than when idle listening or sending/receving data. The radio module also supports different transmit power levels, thereby enabling the use of power-optimized communication algorithms.

To save energy, both the processor and the radio module should be in sleep mode when not transmitting or receiving data. A WINS node sending data with a transmit power of 0.12 mW has an overall power consumption of about 771.1 mW (see Table 1). The overall power consumption increases to 1080.5 mW for a transmit power of 36.3 mW. On the other hand, when the radio module operates in receive (Rx) mode, the WINS node consumes about 751.6 mW.

The amount of *energy* spent to send or receive a given number of bits is proportional to the transmission time and hence to the transmission rate. Given the 100 kbit/s transmission rate of the WINS radio module, the transmission of one bit of data requires an energy of between 7.71 μJ and 10.8 μJ on the sending node, and 7.52 μJ on the receiving node, respectively. The overall energy cost for transmitting (i.e. sending *and* receiving) a single bit of data ranges from 15.2 μJ to 18.3 μJ, whereby the exact value depends on the transmit power level.

5.3 Energy Analysis of Kerberos

The Kerberos protocol, as described in Section 3, requires to send four messages between entity A, entity B, and the trusted third party T. In order to determine the size of the messages shown in Figure 1, we assume that node identifiers (node IDs) and timestamps consist of 64 bits, whereas the session key has a length of 128 bits. Furthermore, we count 32 bits for the nonce n_A that is part of both the AS_REQ and AS_REP message. The second column of Table 2 shows the number of bits for the different messages.

Message	Length	Blk.	Tot. length	Energy
AS_REQ	160 b	–	160+256 b	6.33–7.62 mJ
AS_REP	672 b	6	768+256 b	15.6–18.8 mJ
AP_REQ	448 b	4	512+256 b	11.7–14.1 mJ
AP_REP	64 b	1	128+256 b	5.85–7.04 mJ
All msgs.	1344 b	11	2592 b	39.5–47.5 mJ

Table 2: Communication energy cost of Kerberos.

However, it must be considered that all messages except AS_REQ are AES-encrypted, and since the AES operates on 128-bit blocks, we have to pad each message until its bitlength is a multiple of 128. Besides the actual payload, each message also contains other information like a protocol ID, a message ID, a checksum as reliable indicator of data integrity[3], as well as low-level (MAC and PHY) headers and footers consisting of network addresses and other bookkeeping data. We assume that this "overhead" contributes 256 bits to a message[4]. The fourth column of Table 2 shows the

[3] The IEEE 802.15.4 (ZigBee) standard uses a message integrity code (MIC), which can be 32, 64, or 128 bits long.

[4] For comparison, the IEEE 802.15.4 standard specifies that a data frame can have a payload of up to 816 bits, and the overhead due to the PHY header and the MAC header and footer, respectively, is between 120 and 248 bits.

total length of each message, and the fifth column lists the required energy for sending and receiving the message with respect to different transmit power levels. It turns out that the overall communication energy cost of the four Kerberos messages ranges from 39.5 to 47.5 mJ, whereby the exact value depends on the transmit power level.

As mentioned before, all messages except AS_REQ are AES-encrypted. The third column of Table 2 specifies the number of 128-bit blocks constituting the payload of each message. In summary, eleven encrypted 128-bit blocks are transferred between the entities A, B, and T in each run of the Kerberos protocol. However, it must be considered that the ticket, which is part of both the AS_REP and the AP_REQ message, is encrypted by T and decrypted by B, i.e. entity A just forwards the encrypted ticket from T to entity B. Therefore, only eight encryption and decryption operations on 128-bit blocks have to be carried out, given that the ticket consists of three 128-bit blocks.

The AES encryption of a single 128-bit block of data using a 128-bit key requires approximately 2109 clock cycles on the StrongARM, whereas the decryption of a single block takes 2367 cycles, and the key scheduling requires some 670 clock cycles[5]. Given a clock frequency of 133 MHz, the corresponding energy values are 5.7 μJ (for encryption), 6.4 μJ (decryption), and 1.8 μJ (key scheduling). In summary, the encryption and decryption of eight 128-bit blocks requires less than 0.1 mJ altogether.

Protocol	Comp.	Msg. transfer	Total energy
Kerberos	0.1 mJ	39.5–47.5 mJ	39.6–47.6 mJ
ECMQV	51.8 mJ	27.2–32.8 mJ	79.0–84.6 mJ

Table 3: Energy analysis of Kerberos and ECMQV.

Putting it all together, we find that the overall energy cost (i.e. computation and communication energy) of key establishment according to the Kerberos protocol is between 39.6 mJ and 47.6 mJ (see Table 3). The communication energy constitutes the lion's share of the overall energy, whereas the computation energy is virtually negligible.

Other Factors Impacting the Communication Energy

A fundamental property of the Kerberos key establishment is that entity A obtains a session key from the trusted third party T through the AS_REQ/AS_REP message exchange. The actual energy requirements of Kerberos also depend on whether the energy-rich base station or a conventional (energy-constrained) sensor node implements the trusted third party. In the former case, assuming that the base station has an unlimited energy store, the energy for receiving the AS_REQ message and for sending the AS_REP message does not need to be considered, which reduces the overall energy cost by almost 30% (from 39.6-47.6 mJ to 28.6-33.4 mJ). On the other hand, there exist also protocols where a conventional sensor node plays the role of the trusted third party, and, in this case, the energy for receiving/sending the AS_REQ/AS_REP message must be taken into account. An example for such a protocol is PIKE [5], which combines the basic ideas of random key pre-distribution and Kerberos-like key establishment.

# nodes	Total energy
0	39.6–47.6 mJ
1	61.5–73.9 mJ
2	83.4–100.3 mJ
3	105.3–126.7 mJ
4	127.2–153.0 mJ
5	149.0–179.4 mJ
6	170.9–205.7 mJ

Table 4: Total energy of Kerberos depending on the number of intermediary nodes between A and T.

Our energy analysis of the Kerberos protocol shown in Table 3 is based on the assumption that entity A can directly send/receive messages to/from the third party T, i.e. A is within communication distance of T and vice versa. While this is a reasonable assumption for small sensor networks, it is almost never the case in a large network where the sensor nodes may be located far away from the base station. The communication energy cost of Kerberos depends not only on the transmit power level, but also on the number of intermediary nodes between A and T. Multi-hop communication between A and T increases overall energy consumption since any intermediary node has to forward (i.e. receive and retransmit) the message to its neighbor located on the route to the final destination. Table 4 summarizes the overall energy cost of Kerberos key establishment depending on the number of intermediary nodes between A and T (we assume that A and B have no intermediary nodes).

According to the results in Table 3 and 4, our lightweight Kerberos protocol is more energy efficient than ECMQV when A is within direct communication distance to T or when at most one intermediary node lies between them. On the other hand, ECMQV requires less energy than Kerberos if the communication between A and T passes through two or more hops, which is almost always the case in a large sensor network.

Other Factors Impacting the Computation Energy

State-of-the-art RF modules compliant to the IEEE 802.15.4 standard feature an AES hardware engine to reduce the energy consumption of encryption/decryption operations. Unfortunately, performing AES in hardware instead of software has virtually no impact on the overall energy of the Kerberos protocol since the communication energy is orders of magnitude higher than the computation energy.

5.4 Energy Analysis of ECDH/ECMQV

The conventional (unauthenticated) ECDH protocol is a two-pass protocol, which means that two message transfers are necessary to establish a shared secret key. Each run of the ECMQV protocol also requires the transmission of two messages, provided that the public parts of the static keys have already been exchanged (see Section 4). The payload of these messages consists of the x and y coordinate of the elliptic curve point that represents the public part of the ephemeral key. Both coordinates amount to 512 bits when using a 256-bit field as underlying algebraic structure[6]. We

[5] All cycle counts have been evaluated by simulations with SimIt-ARM [18], a cycle-accurate StrongARM simulator.

[6] We use a 256-bit field to evaluate the energy requirements of the ECMQV protocol since a properly selected 256-bit elliptic curve system provides the same level of security as a symmetric algorithm like AES with a 128-bit key [14].

also include the identities (IDs) of the two entities A and B involved in the key exchange, which increases the payload of each message to 640 bits. In addition, we count for each message an overhead of 256 bits for protocol ID, message ID, checksum, and low-level (i.e. MAC and PHY) headers and footers, respectively. Thus, we have to transmit 896 bits from entity A to entity B, and the same amount in the opposite direction, which results in an overall communication energy of between 27.2 mJ and 32.8 mJ for both messages (the exact value depends on the transmit power level).

Key exchange protocols using public-key primitives, such as ECDH and ECMQV, are very computation-intensive in relation to Kerberos. Each of the two entities participating in ECDH key exchange has to compute two standard point multiplications of the form $k \cdot P$, while ECMQV requires each entity to compute a standard point multiplication (for the generation of the ephemeral public key) and a multiple point multiplication of the form $k \cdot P + l \cdot Q$ (to derive the shared secret according to Algorithm 1). The computation time and energy of the point multiplications carried out in ECDH/ECMQV depends heavily on the group order, and hence on the order of the underlying finite field.

We have selected a 256-bit prime field for the evaluation of ECMQV in order to match the security level of Kerberos with 128-bit AES encryption. More precisely, we have used the prime field GF(p) defined by the generalized-Mersenne prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, in combination with the elliptic curve P-256 as specified in the NIST standard 186-2 [14]. Our implementation performs the multiplication of field elements and the modular reduction according to Algorithm 2.10 and 2.29 in [8]. Furthermore, we have used Algorithm 3.21 and 3.22 to implement the doubling and the addition of points, respectively.

A point multiplication $k \cdot P$ on the P-256 curve requires to carry out 256 doublings and 85 additions on basis of the double-and-add method with NAF-representation of k. Our simulation results show that the StrongARM executes a point multiplication in about $4.25 \cdot 10^6$ clock cycles. The multiple point multiplication (Algorithm 2) is 22% slower ($5.31 \cdot 10^6$ clock cycles) since 128 point additions have to be performed on average when using the JSF. In summary, the energy consumption of a point multiplication $k \cdot P$ is roughly 11.5 mJ, whereas a multiple point multiplication $k \cdot P + l \cdot Q$ consumes circa 14.4 mJ.

Table 3 shows the energy characteristics of the ECMQV protocol. Each entity has to perform a point multiplication and a multiple point multiplication, resulting in an overall computation energy cost of 51.8 mJ. The communication energy cost of ECMQV is only about one-half to two-third of the computation energy cost.

Other Factors Impacting the Communication Energy

The communication energy cost of ECDH/ECMQV can be reduced by applying a technique called *point compression* [2]. Point compression allows to represent a point using the minimum possible number of bits, which is attractive for sensor networks since the transmission of data is relatively expensive. A straightforward representation of a point P on an elliptic curve over a prime field GF(p) requires m bits for each the x and the y coordinate of P, resulting in $2m$ bits altogether, whereby $m = \lceil \log_2(p) \rceil$. The number of bits can be reduced to $m + 1$ since, for a given x coordinate, the Weierstraß equation (Equation 1 in Appendix B) is quadratic in y and has at most two solutions. Consequently, a single bit is sufficient to specify the y coordinate of a point on the curve. Decompressing the point P, i.e. recovering the y coordinate from x and the extra bit, requires to solve a quadratic equation modulo p, which can be done according to Algorithm II.8 in [2].

Unfortunately, the recovery of the y coordinate is highly computation-intensive, and therefore a potential saving in communication energy is, in part, nullified by an additional demand for computation energy. According to our experimental results, point compression does not allow to reduce the overall energy cost of ECDH/ECMQV key exchange by more than 10%.

Other Factors Impacting the Computation Energy

The execution time and energy requirements of both ECDH and ECMQV can be substantially improved by applying an advanced technique for (multiple) point multiplication. For example, window methods or comb methods are faster than the double-and-add method because they need fewer point additions and/or doublings [8]. However, all these advanced techniques have in common that they require pre-computation and storage of multiples of the base point P, which can pose a problem if memory resources are at a premium. An in-depth study of window/comb methods for point multiplication is outside of the scope of this paper. We refer the interested reader to [8] and the references therein.

6. CONCLUSIONS

In this paper we analyzed and compared the energy cost of Kerberos-like key establishment based on AES encryption and a variant of the Diffie-Hellman key exchange (ECMQV) using elliptic curve cryptography. We found that the overall energy cost of the former is between 39.6 and 47.6 mJ, while the latter requires an energy of between 79.0 and 84.6 mJ (the exact value depends on the transmit power level). Consequently, the energy consumption of ECMQV and Kerberos differs by a factor of between 1.78 (for high transmit power) and 2.0 (for low transmit power). Our analysis also shows that the energy consumption of Kerberos is dominated by the message transfers, whereas the energy required for the encryption of data is negligible (see Table 3). In ECMQV key exchange, on the other hand, the computation of cryptographic primitives constitutes approximately 63% of the overall energy consumption. All these results are based on the energy characteristics of the WINS sensor node.

Our results differ significantly from the results of Hodjat and Verbauwhede [9], who found that ECDH key exchange may require between one and two orders of magnitude more energy than Kerberos with AES encryption. The difference between our results and the results in [9] can be explained by the progress in the area of efficient implementation of elliptic curve cryptography in recent years. It can be expected that advances in elliptic curve cryptography will further narrow the energy gap between public-key and secret-key protocols in the future. Thus, we conclude that security protocols using public-key cryptography should no longer be considered prohibitively expensive in terms of energy consumption.

Another important result of our work is that the (relative) energy efficiency of key establishment protocols depends on the size and of a sensor network. In Kerberos and similar protocols (e.g. SPINS), the sensor node initiating the key establishment process needs to communicate with a trusted third party (typically the base station) in order to obtain a session key. This communication is usually multi-hop with other nodes working as routers, especially in large sensor networks. Unfortunately, multi-hop communication is very costly in terms of energy, making Kerberos unattractive in relation to ECMQV. According to our results, ECMQV requires *less* energy than the Kerberos protocol if the node requesting the session key is two or more hops away from the trusted third party.

7. ACKNOWLEDGEMENTS

The authors are grateful to Erkay Savaş and Ingrid Verbauwhede for valuable comments on a draft version of this paper.

8. REFERENCES

[1] J. R. Agre, L. P. Clare, G. J. Pottie, and N. P. Romanov. Development platform for self-organizing wireless sensor networks. In *Unattended Ground Sensor Technologies and Applications*, vol. 3713 of *Proceedings of SPIE*, pp. 257–268. SPIE, 1999.

[2] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.

[3] I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in Elliptic Curve Cryptography.* Cambridge University Press, 2005.

[4] D. W. Carman, P. S. Kruus, and B. J. Matt. Constraints and Approaches for Distributed Sensor Network Security. Technical Report #00-010, NAI Labs, Network Associates, Inc., Glenwood, MD, USA, Sept. 2000.

[5] H. Chan and A. Perrig. PIKE: Peer intermediaries for key establishment in sensor networks. In *Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM 2005)*, vol. 1, pp. 524–535. IEEE, 2005.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov. 1976.

[7] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pp. 41–47. ACM Press, 2002.

[8] D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Verlag, 2004.

[9] A. Hodjat and I. M. Verbauwhede. The energy cost of secrets in ad-hoc networks. In *Proceedings of the 5th IEEE CAS Workshop on Wireless Communications and Networking.* IEEE, Sept. 2002.

[10] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pp. 162–175. ACM Press, 2004.

[11] J. T. Kohl and B. C. Neuman. The Kerberos Network Authentication Service (Version 5). Internet Engineering Task Force, Networking Group, Internet Draft RFC 1510, Sept. 1993.

[12] L. E. Law, A. J. Menezes, M. Qu, J. A. Solinas, and S. A. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, Mar. 2003.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[14] National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). Federal Information Processing Standards (FIPS) Publication 186-2, Feb. 2000.

[15] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, Dec. 1978.

[16] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pp. 189–199. ACM Press, 2001.

[17] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha. Analyzing the energy consumption of security protocols. In *Proceedings of the 8th International Symposium on Low Power Electronics and Design (ISLPED 2003)*, pp. 30–35. ACM Press, 2003.

[18] W. Qin. SimIt-ARM (Release 2.1). Available for download at `http://simit-arm.sourceforge.net`, 2002.

[19] C. S. Raghavendra, K. M. Sivalingam, and T. F. Znati. *Wireless Sensor Networks.* Kluwer Academic Publishers, 2004.

[20] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, Mar. 2002.

[21] A. Sinha and A. P. Chandrakasan. JouleTrack - A web based tool for software energy profiling. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pp. 220–225. ACM Press, 2001.

[22] J. A. Solinas. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Waterloo, Canada, 2001.

[23] E. G. Straus. Addition chains of vectors. *American Mathematical Monthly*, 71(7):806–808, Aug./Sept. 1964.

[24] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. Chang Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communication (PerCom 2005)*, pp. 324–328. IEEE Computer Society Press, 2005.

APPENDIX

A. DETAILED DESCRIPTION OF LIGHTWEIGHT KERBEROS

In this section we outline how our lightweight Kerberos protocol produces a session key and give a brief description of the messages sent between the entities. Let entity A be the initiator of the key establishment process, and B shall be the entity with which A wants to set up a unique session key. We assume that both A and B possess a long-term key with T, but do not initially share a secret key with each other. Figure 1 illustrates the messages transferred between A, B, and T in security protocol notation. For clarity, the figure shows a simplified version of the messages to allow focus on cryptographic aspects (several non-cryptographic fields, such as network addresses, are omitted).

- The first message is the Authentication Server Request (AS_REQ) message, which is sent in plain text from A to T. This message contains A's own identity, the identity of the entity B for which a shared key is requested, and a randomly generated nonce n_A that will be used to associate reply messages with the matching AS_REQ request and to detect replays.

- Upon receipt of an AS_REQ message, the trusted third party T looks up entities A and B its database, verifies that they are authorized to establish a session key, and fetches the corresponding long-term keys k_{AT} and k_{BT}. Then, T generates a new random session key k_{AB} to be shared by A and B and embeds it into a so-called *ticket*. Besides the session key, the ticket also contains the identity of the entity requesting the session key (A in our example), and the ticket's validity period (lifetime), consisting of an expiration time t_E and an optional starting time t_S. The ticket is encrypted using the long-term key k_{BT} only known by T and B, which guarantees that nobody else can read or change the identity of A specified within the ticket.

Next, T assembles a response, the AS_REP message, consisting of the ticket for A to present to B, the session key k_{AB}, the assigned expiration time t_E after which the session key is no longer valid, the identity of the entity for which the ticket was created (B in our example), and the nonce n_A from the AS_REQ message. All elements except the ticket are encrypted with the long-term key k_{AT} shared between T and A (in Kerberos version 4 also the ticket is encrypted in k_{AT}). The AS_REP message provides entity A with the session key k_{AB} and the ticket[7] to forward to B.

- After reception of the AS_REP response, entity A uses the long-term key k_{AT} to decrypt the non-ticket portion of the message, thereby obtaining the session key k_{AB} and other information as specified above. Entity A verifies that the received nonce matches the nonce it supplied in the AS_REQ message (to detect replays) and that the current time is within the lifetime of the session key. Furthermore, entity A checks whether the ticket was actually created for the desired communication partner (entity B in our example) by verifying the identity decrypted from the AS_REP message. When everything matches, entity A assumes that it possesses a valid session key and ticket for entity B.

  In the third message, the AP_REQ (Application Request) message, entity A transfers the ticket together with a so-called *authenticator* to B. The authenticator is generated by entity A and contains A's own identity as well as a fresh timestamp t_A, both encrypted in the session key k_{AB} obtained from the trusted third party T. In Kerberos, the purpose of the authenticator is twofold. First, the authenticator proves that entity A has knowledge of the secret session key k_{AB}, which in turn, proves A's identity[8] and entitles it to use the ticket. Second, because the authenticator contains a timestamp, it ensures that every AP_REQ message is unique (i.e. the authenticator serves to thwart replay attacks).

- Having received the AP_REQ message, entity B first decrypts the ticket (using the long-term key k_{BT} it shares with T) and extracts the session key k_{AB}, the identity of the entity to which the ticket was issued (A in our example), as well as the ticket's expiration time t_E. Now entity B uses the session key k_{AB} from the ticket to decrypt the authenticator and compares the information in the ticket with that in the authenticator. In particular, B verifies that the identity fields obtained from the ticket and the authenticator match, that the timestamp t_A is still valid, and that B's local time is within the lifetime specified in the ticket. If all checks pass, entity B may be reasonably assured that A is in fact the entity named in the ticket, i.e. B now considers A as properly authenticated.

  Mutual authentication requires that entity B proves its identity too. Entity A can easily verify B's identity by requesting that B sends something back that assures that B has access to the session key k_{AB}. The Application Reply (AP_REP) message simply consists of the timestamp[9] encrypted in the session key k_{AB}. After having received and decrypted the AP_REP message, entity A verifies that the timestamp is the same one it sent in the AP_REQ message. This ensures A that the session key k_{AB} has been successfully transmitted to B, since k_{AB} was needed to produce the AP_REP message. Furthermore, the AP_REP message enables A to verify that B really is B, since only B can extract the session key from the ticket.

At the end of this message exchange, entity B is certain that, according to Kerberos, entity A is who it claims to be. Since mutual authentication occurs, A is also convinced that B is authentic. Moreover, A and B share a key which no one else knows. In Kerberos, both the ticket and the authenticator are encrypted, but in different keys. The ticket is generated by the trusted third party T, encrypted in the long-term key shared between T and the entity to which the ticket will be presented (B in our example), and valid for a certain period. On the other hand, the authenticator is built by the requesting entity A itself, encrypted in the session key k_{AB}, and can be used only once.

B. DIFFIE-HELLMAN KEY EXCHANGE

In its simplest form, the Diffie-Hellman key exchange is carried out in a multiplicative group of integers modulo a large prime p, in the following denoted as $\mathbb{Z}_p^*$. Let g with $2 \leq g \leq p-2$ be a *generator* of $\mathbb{Z}_p^*$. We assume that p, g are publicly known to every entity of the network. Whenever two entities, A and B, wish to establish a shared secret key k, they accomplish the following actions [13]:

- Entity A chooses a random secret a with $1 \leq a \leq p-2$, computes $s = g^a \bmod p$, and sends s to entity B.
- Entity B chooses a random secret b with $1 \leq b \leq p-2$, computes $t = g^b \bmod p$, and sends t to entity A.
- When B receives s from A, it computes the shared key as $k = s^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$.
- When A receives t from B, it computes the shared key as $k = t^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p$.

In summary, entity A derives the shared key k from its own secret value a and the value t generated by B, while entity B uses its own secret value b and the value s generated by A to calculate k. Entity A and B possess the same secret key k since $t^a \bmod p = s^b \bmod p = g^{ab} \bmod p = k$. An adversary knows p and g, and may be able to obtain the values s and t by eavesdropping on the communication between A

[7] The ticket is not sent directly to the B, but is instead sent to A. However, A can not read or modify the ticket as it is encrypted in the key k_{BT} only known to T and B.

[8] The ticket by itself is insufficient for authentication since an attacker could simply eavesdrop on the communication and replay a ticket in order to impersonate entity A. Therefore, the authenticator (rather than the ticket) is what really proves A's identity, since only a legitimate entity possessing the session key can generate a valid authenticator.

[9] In version 4 of the Kerberos protocol, the AP_REP message contains the timestamp value plus one.

and B. However, knowledge of p, g, s, t is insufficient to derive the secret key k. On the other hand, if an adversary would be able to determine a from $s = g^a \bmod p$ (or b from $t = g^b \bmod p$), he could easily calculate k in the same way as entity A (or B) does. But finding a given s, g, and p is an instance of the discrete logarithm problem (DLP), which is computationally infeasible when the prime p is sufficiently large (e.g. $p \geq 1024$ bits). The Diffie-Hellman protocol can be carried out in any group in which the DLP is hard and the group operation is efficient, e.g. in the group of points defined by an elliptic curve over a finite field.

Elliptic Curve Diffie-Hellman (ECDH)

Performing a modular exponentiation on operands with a length of 1024 bits, such as required for Diffie-Hellman key exchange in the group $\mathbb{Z}_p^*$, is very computation-intensive and, hence, energy consuming. The main advantage of embedding the Diffie-Hellman protocol into the group of points on an elliptic curve is that taking the discrete logarithm in such groups is much harder than in $\mathbb{Z}_p^*$. Therefore, a desired level of security can be attained with much smaller group orders (e.g. 160 to 256 bits), which makes the elliptic curve Diffie-Hellman (ECDH) key exchange particulary attractive for sensor nodes and similar devices with restricted memory resources and low-bandwidth network connectivity.

An elliptic curve can be constructed over different algebraic structures like a ring or a field. However, it is common practice in elliptic curve cryptography to use either a prime field GF(p) or a binary extension field GF(2^m) since these two field types are recommended by several standards, including the NIST standard FIPS 186-2 [14]. Implementation results from [8] indicate that prime fields perform better in software than binary extension fields. Therefore, we only consider elliptic curve systems over prime fields in the rest of this paper.

Formally, an elliptic curve over a prime field GF(p) can be defined by a Weierstraß equation of the form

$$y^2 = x^3 + \alpha x + \beta \qquad (1)$$

where $\alpha, \beta \in \mathrm{GF}(p)$ and $4\alpha^3 + 27\beta^2 \neq 0 \bmod p$ [2]. A tuple $(x, y) \in \mathrm{GF}(p) \times \mathrm{GF}(p)$ satisfying Equation (1) is a point on the curve. The set of all points, together with a special point called the *point at infinity* $\mathcal{O}$, allows to construct an abelian group, whereby the group operation is the addition of points and $\mathcal{O}$ is the neutral element. A point addition is performed through arithmetic operations in the underlying prime field according to well-defined formulae (see [2, 8] for further details). The basic building block of virtually all elliptic curve cryptosystems is a so-called *scalar multiplication* or *point multiplication*, which is simply an operation of the form $k \cdot P$, whereby k is an integer and P is a point on the curve. Calculating $k \cdot P$ is nothing else than adding the point P exactly $k - 1$ times to itself, which results in another point R on the curve[10]. Thus, a scalar multiplication can be accomplished by a sequence of point additions and doublings, respectively. The inverse operation, i.e. to recover k when the points P and $R = k \cdot P$ are given, is an instance of the elliptic curve discrete logarithm problem (ECDLP).

The ECDH protocol works very similar to the "traditional" Diffie-Hellman key exchange; the main difference is that the group $\mathbb{Z}_p^*$ is replaced by the group of points on an elliptic curve. In what follows, let $\mathbb{E}$ be an elliptic curve group of order n, and G shall be a point on the curve, i.e. $G \in \mathbb{E}$. For sake of simplicity we assume that the order n is prime, which means that $\mathbb{E}$ is cyclic and G is a generator of $\mathbb{E}$. Furthermore, we assume the domain parameters p, α, β, n, and G are publicly known to every entity of the network. Let A and B be two entities wishing to establish a shared key. First, entity A chooses a random secret number a with $2 \leq a \leq n - 2$, calculates $S = a \cdot G$, and sends S to entity B. Entity B also chooses a random secret number b in the range of $[\,2, n - 2\,]$, calculates $T = b \cdot G$, and sends T to entity A. Entity A can now compute the shared key as $K = a \cdot T = a \cdot b \cdot G$ and entity B is able to compute $K = b \cdot S = b \cdot a \cdot G$. Both entities have agreed on the same key since $\mathbb{E}$ is an abelian group, i.e. $a \cdot b \cdot G = b \cdot a \cdot G$.

In summary, the ECDH key exchange requires to perform four scalar multiplications and to send two messages.

[10] Scalar multiplication in an additive group is equivalent to exponentiation in a multiplicative group like $\mathbb{Z}_p^*$. Both are performed by repeatedly applying the group operation (addition or multiplication) to a group element.