Cryptographic Key Exchange in IPv6-Based Low Power, Lossy Networks

Panagiotis Ilia, George Oikonomou, and Theo Tryfonas

Cryptography Group, Faculty of Engineering, University of Bristol,
Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK
panagiotis.ilia.2011@my.bristol.ac.uk,
{g.oikonomou,theo.tryfonas}@bristol.ac.uk

Abstract. The IEEE 802.15.4 standard for low-power radio communications defines techniques for the encryption of layer 2 network frames but does not discuss methods for the establishment of encryption keys. The constrained nature of wireless sensor devices poses many challenges to the process of key establishment. In this paper, we investigate whether any of the existing key exchange techniques developed for traditional, application-centric wireless sensor networks (WSN) are applicable and viable for IPv6 over Low power Wireless Personal Area Networks (6LoWPANs). We use Elliptic Curve Cryptography (ECC) to implement and apply the Elliptic Curve Diffie Hellman (ECDH) key exchange algorithm and we build a mechanism for generating, storing and managing secret keys. The mechanism has been implemented for the Contiki open source embedded operating system. We use the Cooja simulator to investigate a simple network consisting of two sensor nodes in order to identify the characteristics of the ECDH technique. We also simulate a larger network to examine the solution's performance and scalability. Based on those results, we draw our conclusions, highlight open issues and suggest further work.

Keywords: 6LoWPAN, Key Exchange, ECC, ECDH.

1 Introduction

Wireless Sensor Networks (WSNs) consist of a large number of autonomous devices that cooperate to collect important data and send them through wireless communication channels to a base station or a data centre. Every node mainly consists of a microcontroller, a memory unit, a transceiver, a power source and one or more sensing elements. Due to their nature, wireless sensors are very constrained in terms of available RAM, speed of computation, network bandwidth and battery lifetime.

In 2003, the IEEE published the first version of IEEE 802.15.4, a specification for the physical and link layer operation for low-power radio communication. Initial research efforts suggested that TCP/IP was not viable for WSNs and that bespoke, application-centric network stacks were more suitable [1, 2]. However,

L. Cavallaro and D. Gollmann (Eds.): WISTP 2013, LNCS 7886, pp. 34–49, 2013.

the release of uIP demonstrated that standards-compliant TCP/IP stacks for embedded devices are viable [3]. Subsequently, it was shown that a TCP/IP-based WSN could outperform traditional, application-centric network designs [4]. As a result, a series of Internet specifications have been suggested for the transmission and routing of datagrams with IPv6 over Low power Wireless Personal Area Networks (6LoWPANs) [5].

With 6LoWPAN, WSN nodes with IEEE 802.15.4 radio transceivers are directly accessible from the Internet and are exposed to a host of security threats. Modern sensor devices are often equipped with an encryption/decryption coprocessor and link layer frames can be transmitted encrypted with the 128-bit Advanced Encryption Standard (AES) algorithm. However, there are challenges associated with key management and the process of key exchange in 6LoWPANs is not trivial. On many occasions, keys are individually pre-loaded to the nodes, which can be characterized as an important security vulnerability.

In this paper, we investigate whether any of the existing techniques for dynamic generation and exchange of cryptographic keys are applicable and can be adopted for 6LoWPANs. We argue that techniques based on Elliptic Curve Cryptography (ECC) [6] are very promising and we implement an Elliptic Curve Diffie Hellman (ECDH) shared key generation and establishment algorithm. This mechanism is also responsible for the management of existing encryption keys, searching and returning them to the link layer when requested, or starting the key exchange process if two neighbouring nodes do not have a shared secret key. It additionally handles key storage, expiration and replacement.

We implemented the key management mechanism and ECDH algorithm for the Contiki embedded operating system[1] and we evaluated it in terms of applicability, viability and scalability with network size and density. We evaluate memory footprint for a single node as well as how the number of stored keys affects network scalability. In addition, we use the Cooja simulator to conduct several simulations and assess node energy consumption, network lifetime and performance.

2 Background

Key management schemes proposed for WSNs are divided mainly into two different categories: i) symmetric key schemes where the keys are either pre-installed or assigned by a trusted party and ii) schemes based on Public Key Cryptography (PKC).

2.1 Symmetric Key Schemes

Compared to PKC, symmetric key schemes have the advantage that they are less computationally intensive, requiring fewer micro-controller instruction cycles to perform the calculations required for encryption and decryption.

[1] `http://www.contiki-os.org`

Keys are pre-installed during the network's deployment and initialization phase and neighbouring nodes discover and establish a shared key during the network formation phase [7].

Among existing efforts is a key exchange approach which requires the existence of a trusted party in the WSN, acting as a Key Distribution Centre (KDC), a role which can be assumed by a Base Station (BS) [8]. In this scheme the KDC must establish a secure, single-hop communication channel with every node of the network, in order to deliver the secret keys. The trusted entity-based key exchange scheme cannot be applied in 6LoWPANs because the assumption that all nodes are within a single hop of the base station does not always hold true: The aim of 6LoWPAN and related specifications is to facilitate the formation of multi-hop networks [9].

A different approach relies on each node having a pre-installed set of keys chosen from a key pool. These schemes can be either deterministic or probabilistic. Under deterministic schemes, every node is capable of establishing a pair-wise key with all its neighbors. One method that stands out is the one proposed in [10], whereby every two nodes in the network share exactly one common key. According to Bechit et al., deterministic schemes do not scale well with network size [11] and are thus unsuitable for 6LoWPANs where scalability is a desirable feature.

Under probabilistic schemes, a common key is present between two neighbors with some probability. For instance, under the scheme documented in [12], a small subset of k keys is chosen randomly out of a large key pool S. Every network node exchanges the identifiers of its keys with its neighbors and, if a common key exists, it is used as their session secret key. A more contemporary probabilistic scheme is the one proposed in [11]. Because of the probabilistic nature of such schemes, many pairs of nodes do not share a common secret key and thus, they try to find a secure routing path through their neighbors in order establish it. If the connectivity of the network graph is not high, it is possible that the network becomes partitioned into sub-networks and thus it may be impossible to discover paths for key establishment between the two parts. Since full network connectivity is not guaranteed, such schemes are unsuitable for 6LoWPANs [8]. Additionally, if a single node is compromised a large subset of the global key pool may be revealed to the attacker [11]. However, because keys are pre-installed, revocation and replacement is not trivial.

2.2 Public Key Cryptography

Public key schemes are more demanding in terms of computation and energy consumption than symmetric key schemes. However, Zhang and Varadharajan argue that public key schemes provide a higher level of security, they scale better with network size and they have lower storage requirements [7].

The classic Diffie Hellman algorithm uses keys of very large size and does not provide any authentication mechanism, unless used alongside other protocols. It is reported that the RSA encryption algorithm is viable in WSNs despite using a large key [13–15]. The advantage of RSA over Diffie Hellman is that it can

provide both key exchange and authentication with a single pair of keys (public-private). However, the processes of key generation and encryption (for secret key establishment) are still very slow and energy consuming.

Elliptic Curve Cryptography (ECC) is a very attractive solution for 6LoWPANs, since key length is considerably smaller than that used by other traditional PKC schemes. Consequently, according to NIST recommendations, the strength of a 160-bit ECC key is equivalent to a 1024-bit RSA key [16].

Reportedly RSA and ECC cryptosystems with 1024-bits and 160-bits key size respectively have been implemented on MICA motes and PKC is a feasible security solution for sensor networks [13]. Additionally, software implementations of RSA and ECC public-key algorithms exist for Atmel AVR Atmega128 microcontrollers [14], which is a hardware platform commonly encountered in 6LoWPAN deployments. It is observed that the 160-bits ECC is not only much faster than the equivalent 1024-bits RSA, but it also uses less memory for data and code hosting. The authors of that work extend their research by implementing RSA key exchange with mutual authentication as well as ECDH with ECDSA, between two non-trusted parties [15]. They discuss a simplified and lightweight Secure Sockets Layer (SSL) protocol in order to allow sensor nodes to perform a handshake and subsequently to negotiate and establish a secret key. Energy consumption for a full handshake with ECC is four times lower than with RSA.

Bianchi et al. introduce the asymmetric scheme of Identity Based Cryptography (IBC), which is based on bilinear pairings on elliptic curves, as a promising key exchange scheme for WSNs [17]. The IPC scheme was subsequently adopted for IP-based WSNs [18]. The fundamental idea of IBC is that every string, like the identity of each node (ID), can be used as a valid public key and thus the use of large certificates for authentication is avoided. By using this approach, nodes are able to establish a common secret key without any communication. However, the main drawback stems from the fact that private keys are computed only by the trusted authority (TA) by using the ID of the each node and its secret key. If a particular key is leaked, the TA must pick a new secret key and start a re-keying phase.

3 Implementation of the Key Exchange Technique

We implemented the ECDH key exchange technique for the Contiki OS, which is a portable and lightweight operating system, specifically designed for use by devices with limited resources. The Contiki OS supports a full TCP/IP network stack, including support for a host of standard internet protocols, such as IPv6, UDP, TCP, ICMP and HTTP. It also implements the 6LoWPAN adaptation layer as defined in IETF's Request For Comments (RFC) 4944 [5] and the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [9].

3.1 Link Layer Frame and Framer

Our proposed key exchange solution is implemented as a daemon process and underpins the network stack's secure link layer frame transmission by establishing

secret keys for symmetric encryption between single-hop neighbor nodes. Contiki's implementation of IEEE 802.15.4 Medium Access Control (MAC) frame generation and parsing does not currently support the security header specified by the IEEE 802.15.4 standard. We have modified the respective code module to support the generation and parsing of the security header in a standards-compliant fashion. If the application determines that the MAC frame needs to be secure (by defining the security bit in the frame control field), the security header is appended to the address field of the MAC header, prior to the data payload.

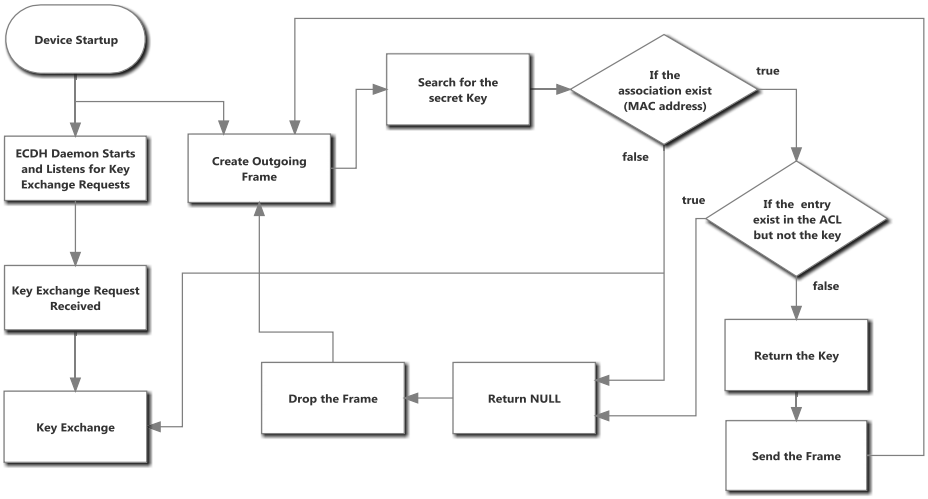

Fig. 1. Flowchart for Secret Key Search, Frame Creation and Transmission

The link layer frame generation module accepts outgoing frames from the layer 2 driver, adds the MAC header at the beginning of each frame and delivers them to the radio transceiver's driver. It also receives incoming frames, parses and removes the MAC header and delivers the payload to the upper layers. Apart from the frame structure, we also modified the link layer framer to fill the values of the Auxiliary Security Header.

The link layer on the transmitting node queries our driver for the existence of a shared key with the intended recipient, identified by the frame's destination MAC address. We search our Access Control List (ACL) for an entry matching the destination MAC address and, if an entry exists, we return the established key as shown in Fig. 1. If the association is not in the ACL the frame is dropped due to the lack of security guarantees and an internal transparent process for key exchange starts.

Fig. 2 illustrates a state transition diagram for our key exchange daemon process. A device will spend most of its time in the *Listening* state. When the network stack attempts to transmit a frame, the node will transition to the

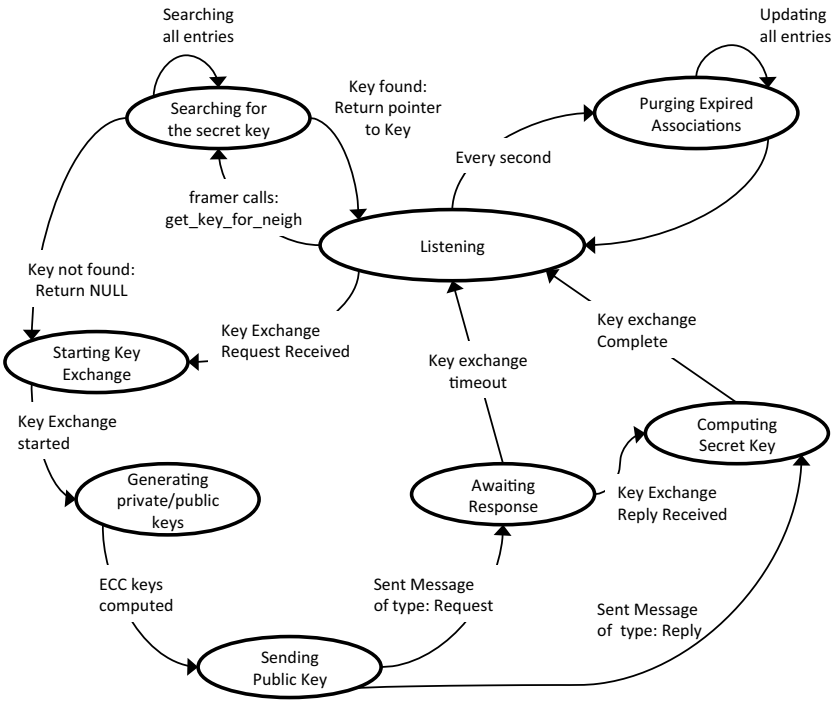

Fig. 2. State Transition Diagram for the Process of Key Exchange

Searching for the secret key state. Depending on the outcome of the key search, the daemon will either return a pointer to the shared key and revert to the *Listening* state or it will move to the *Starting Key Exchange* state.

3.2 ECC Implementation

In the current work, we used source code from the ContikiECC project [19] in order to implement the basic elliptic curve operations. The ContikiECC project is a Contiki port of the TinyECC library [20]. It implements functions to handle very large numbers as multiple 8-bit or 16-bit words and provides the basic numerical operations for 8-bit and 16-bit microprocessors. Moreover, it provides a number of elliptic curves of sizes 128, 160 and 192 bits by specifying each curve's parameters and base point. As discussed above, modern sensor hardware platforms provide hardware acceleration for 128-bit AES encryption, hence our decision to use a curve of 128 bits for the construction of secret keys.

From the ContikiECC library, we use the basic large number operations to implement the ECC operations of point addition, point doubling and multiplication by a scalar multiplier. ECC operations are based on the sliding window method which provides more optimised characteristics in comparison to other methods.

In order to implement the Diffie Hellman algorithm over elliptic curves, each node creates an ephemeral private key as a random 128-bit number. By multiplying the private key with the elliptic curve's base point, we compute the node's public key. For the establishment of a shared secret key between two parties, each party multiplies its own private key with the public key received by the node it is negotiating with. Public keys are transmitted as clear text. A new private-public key pair is used for each negotiation.

For the generation of random numbers, we use Contiki's library which provides a platform-independent Application Programming Interface (API). Each platform supported by Contiki provides a hardware-specific Random Number Generator (RNG) implementation, which underpins this API. For instance, the cc2430 and cc2530 System-on-Chip (SoC) platforms provide hardware-based RNG implementations, while other platforms rely on software. In most cases, random bits from the Radio Transceiver's receive path are used to seed the RNG implementation.

3.3 Key Storage and Management

We construct a custom data structure (key association) that links the destination node's MAC address, the established secret key, key lifetime and the state of the key exchange procedure. The information remains stored in the structure until the shared key expires. After the shared secret has been established, the private-public key pair used to generate it is erased.

Every node in the network has a statically pre-allocated ACL table for storing key association data structures. Each key association has a lifetime (in seconds), which is set when the key establishment process is over. Every second, the daemon periodically enters the *Purging Expired Associations* state (Fig. 2) and decrements key lifetimes by one. When an entry's lifetime reaches zero the shared secret is erased, the MAC address is set to all zeros and the entry's state is reset. This releases the association, which can then be allocated for a new key establishment in the future.

At the beginning of the key exchange process the sensor node allocates a free ACL entry to store the new association in relation to the destination's MAC address. A special situation is the case where an entry is not completed but is already allocated, which means that another key exchange process is in progress (with different neighbor). By handling this situation we avoid the re-allocation of an already allocated association and we can support multiple concurrent key establishment negotiations.

3.4 The Key Exchange Process

The Diffie Hellman key exchange daemon is a background process, which remains idle as long as a key exchange is not requested. The process is transparent and application-independent.

The ECDH daemon is triggered if a secret key is requested by the network stack but does not exist in the ACL table. In order to start the key exchange

process, the daemon first allocates a free ACL entry. It then queries the node's Neighbor Discovery (ND) cache to determine the IPv6 address of the destination node. If both steps are successful, the daemon computes the ECC private-public key pair and sends a key exchange request to the destination node over UDP.

We build a simple protocol for the key exchange messages, defining the Protocol Version, Message Type and Payload, which actually is the sender's public key. Reception of a key exchange request also triggers the ECDH daemon. Upon reception of an ECDH message, the receiving node first validates the protocol version and message type, as illustrated in Fig. 3.

If the message type is Request, it means that the sender node is asking for key exchange and sends its public key. Thus, the receiver allocates an ACL entry, creates its own ephemeral ECC keys, sends a Reply message, computes the secret shared key by performing elliptic curve point multiplication and sets the key lifetime. In this case, the daemon enters the states of *Generating Private/Public Keys*, *Sending Public Key*, *Awaiting Response* and *Computing Secret Key* in that order, as shown in the state transition diagram in Fig. 2. On the other hand, if the received message is a key exchange reply, the node searches to find the related ACL association and computes the secret shared key by using the data of the specific association entry and the received public key.

To overcome the situation of an ACL association being permanently allocated because of indefinitely waiting for a key exchange reply message, we set a key-exchange timeout by setting the association's lifetime to a low value during the

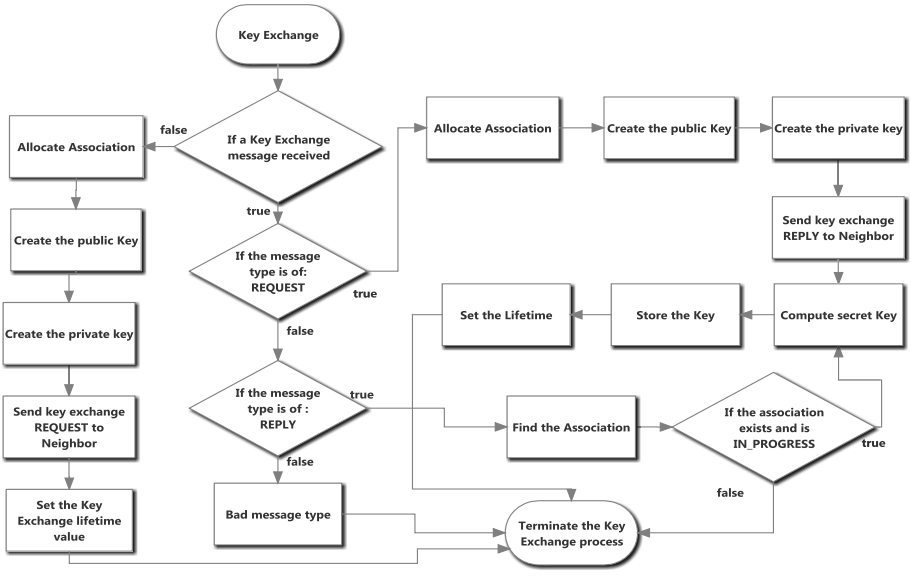

Fig. 3. Flowchart of the Key Exchange Process

negotiation. After the temporary lifetime is initialized, it decreases periodically similarly to the secret key lifetime. If the key-exchange attempt expires, the allocated ACL entry is released and the daemon transitions from the *Awaiting Response* state to *Listening*.

4 Experimental Setup, Results and Analysis

We evaluate the key exchange technique with the Cooja simulator[2], which is distributed with the Contiki OS. Default parameters used for our experiments are provided in Table 1. Some experiments use different configuration parameters, in which case the modifications are clearly discussed in the text. Cooja can emulate motes at the hardware level, allowing precise inspection of system behavior. The sky platform is very commonly used and very well supported by Cooja and for that reason it has been chosen for our experiments.

Table 1. Simulation Configuration

Parameter	Value
Motes	Tmote Sky
TX Range	50m
MAC Layer	IEEE 802.15.4
Radio Access	CSMA
Duty Cycling	ContikiMAC
Max Neighbors	4
ACL Size	4
ACL Entry Lifetime	1000, 1500, ..., 2500 secs
Key Exchange Lifetime	50 seconds

4.1 Memory Requirements

The memory requirements of the key exchange technique are presented in Table 4.1. For implementing the ECDH key exchange method we use the 16-bit mode of ContikiECC's libraries (line *nn* in the table). Moreover, we use the Standards for Efficient Cryptography Group (SECG) standardized elliptic curve *SECP128R1*, by defining curve parameters a and b and its base point. The *ecdh* line relates to the ECDH daemon, which implements the ACL table, provides the key management mechanism and handles the process of key exchange.

Results show that we spend about 7.7 Kb of the device's ROM memory for source code hosting and about 1.1 Kb of RAM for storing curve parameters and ACL associations.

[2] `http://www.contiki-os.org/start.html#start-cooja`

Table 2. Memory and Code Footprints in Bytes

Code Module	ROM	RAM		Overall
		data	bss	
nn	3372	0	0	3372
ecc	2494	0	676	3170
ecdh	1477	10	392	1879
secp128r1	402	0	0	402
Total	7745	10	1068	8823

4.2 Latency, Average Energy Consumption and Network Scalability

The first and simplest experiment consists of only two nodes that communicate for a long time period so that many secret key re-establishments can take place. This experiment investigates energy consumption and computation times required for the calculation of public and secret shared keys as well as for a full key exchange process.

In this example, key exchange always begins with Alice, while Bob is always the node receiving Alice's request and has to reply. The communication is performed properly as long as the lifetime of the secret key has not reached the value of zero. When the secret key expires a new secret key must be established for further communication.

The time needed for the creation of the public key and the computation of the secret key, as well as the overall time for the key exchange process is given in Table 3(a). Table 3(b) presents the energy Alice and Bob consume to create their public key and the shared secret key. It also presents the overall energy consumption for the full key exchange process.

To estimate energy consumption, we use Contiki's energest module. We measure the time each node spent in each of the following three states: i) Micro-Controller Unit (MCU) active, ii) RF listening / receiving (RX), iii) RF transmitting (TX). Since we are simulating sky motes, we then converted these time

Table 3. Average Time and Energy Consumption of the ECDH Key Exchange Process

(a) Average Time Consumption (seconds)

	Public key	Secret key	Key Exchange Process
Alice	8.560	8.547	25.586
Bob	8.457	8.416	16.873

(b) Average Energy Consumption (mJ)

	Public key	Secret Key	Key Exchange Process	
			MCU	MCU + TX + RX
Alice	47.176	47.226	97.021	113.224
Bob	47.165	47.171	94.353	100.758

values to estimated energy consumption based on typical datasheet power levels at an operating voltage of 3.0V. Energy spent for the creation of the public and the secret key is calculated based solely on microcontroller activity. The total for the key exchange also takes into account consumption attributed to the radio transceiver.

In terms of network scalability, default values configure ACLs to hold a maximum of four concurrent key associations. Each increase to the maximum ACL size by one increases the table's memory footprint by 84 bytes, as illustrated in Fig. 4. With the configuration used for our experiments, we could build working firmware with a table size of up to 36 entries before we started getting linker errors.

This does not mean that the network cannot support more than 36 nodes, but that each sensor can support only up to 36 different secret keys at any time. Bearing in mind that keys are only generated between single-hop neighbors, results demonstrate that it is possible to build very dense networks (each node has 30 or more neighbors) without problems.

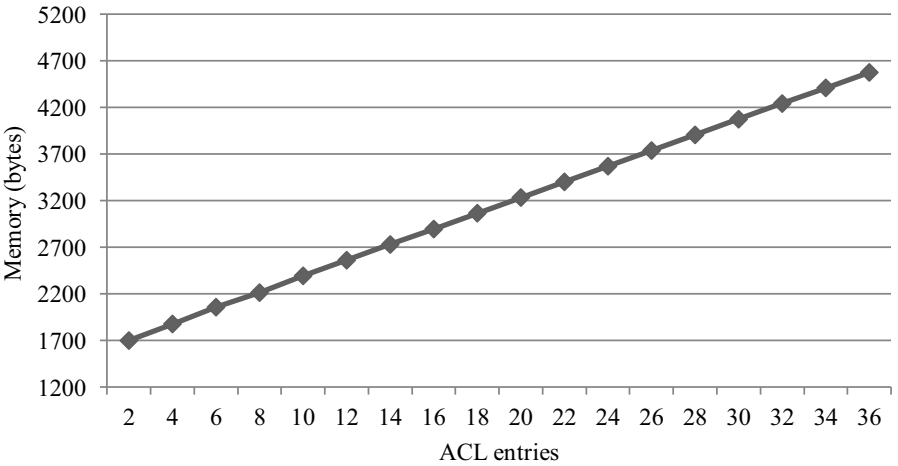

Fig. 4. Scalability with Network Density

4.3 Energy Consumption and Key Lifetime

In this experiment we build a network of ten nodes, where one of them acts as a simple UDP sink node (server) and the rest act as clients. Four of the client nodes are out of the server's transmission range and thus the communication between them and the server is conducted through intermediate routers.

We perform multiple simulation runs, starting with a key lifetime value of 1000 seconds and for each consecutive run the value increases by 500 seconds. Simulations run for a period of two hours so that the keys expire multiple times and many key negotiations take place.

Fig. 5 presents the energy consumption of each network device, for a deployment without key exchange support and for key lifetime values of 1000 and 2500 seconds. The bars in the graph show total energy consumption, with bar portions illustrating consumption attributed to micro-processor activity, RF listening/reception and RF transmission.

Both the microcontroller's and the overall energy consumption of the server device (node ID: 0) is very high in contrast to the estimated energy consumption of the other devices. This happens because the server node has more connections and is required to establish multiple secret keys. Transmission at the server consumes the smallest amount of energy in contrast to the other nodes and reception is the highest. This is due to the functionalities the node implements, as the server transmits only its public key during the key exchange process, while the other nodes transmit their keys as well as application layer data.

Nodes acting as internal routers (node IDs 6-9) consume a large amount of energy in relation to the remaining nodes. The energy consumed by its microprocessor was spent not only to establish a secret key with the server, but also with the client nodes it serves. Similarly, the energy consumed by transmission is due to sending its own application layer messages as well as routing application layer messages towards the server.

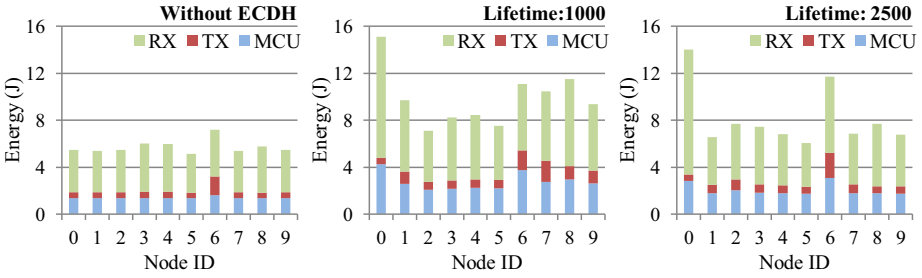

Fig. 5. Average Energy Consumption without ECHD and with ECDH for Key Lifetime Values of 1000 and 2500 Seconds

Results presented in Fig. 6 show that energy consumed by node microprocessors decreases as key lifetime increases. This happens because the keys are valid for a longer time frame and thus, fewer key negotiations are needed. The energy consumption of the server's receiver is fluctuating but remains high, because while the number of the key exchanges decreases, the number of received messages increases and thus the receiver remains busy.

The energy spent by router MCU is lower than the server's MCU energy consumption and higher than the client's average MCU consumption, as it is related to the number of the key exchanges it performs. However, as key lifetime increases, MCU energy consumption decreases.

It is observed that the average energy consumption of the micro-processor of the clients decreases as key lifetime increases. The same phenomenon applies

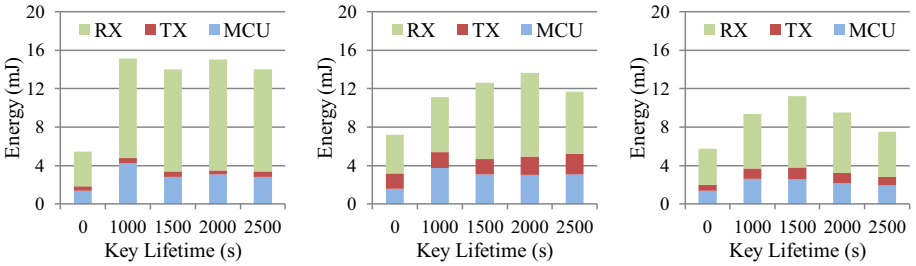

Fig. 6. Energy Consumption vs Key Lifetime.: Server *ID:0* (left), Router *ID:6* (middle); Clients (right).

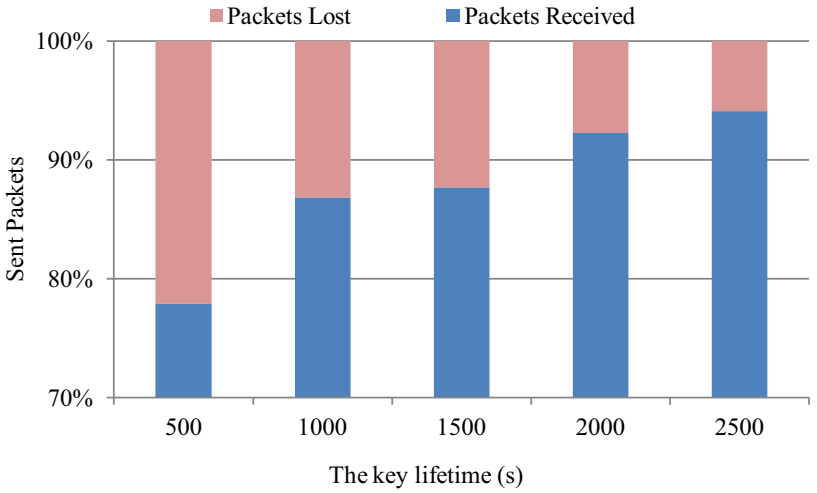

Fig. 7. Packet Loss vs Key Lifetime

to the energy consumed during transmission. However, consumption due to reception increases at the key lifetime of 1500 seconds, for the specific network setup.

4.4 Key Lifetime and Packet Loss

Outgoing packets are dropped at layer 2 when a secret key does not exist in the ACL table and in the case that the key exchange process is in progress. By dropping the outgoing packets the overall performance of the network is affected. However, the existence of the secret key is determined by the lifetime value, which defines whether an entry in the ACL table is valid or not.

To assess how the lifetime affects the network performance we repeat the previous simulations and measure packet loss, with results for various key lifetime values illustrated in Fig. 7. When key lifetime is set at 500 seconds the packets

lost due to the lack of the key are more than 22 percent of the outgoing packets. This happens because key lifetime is short and nodes need to re-negotiate keys relatively frequently. The packet-loss ratio decreases significantly as the key lifetime increases. When the lifetime is set to 1000 and 1500 the packet-loss of the networks is 13% and 12% respectively. Eventually, when the lifetime is set to 2000 and 2500 seconds the packet-loss gets lower than 10% and 5% respectively.

5 Conclusions and Further Work

We developed a software implementation of the ECDH key exchange algorithm in order to assess its applicability and viability for 6LoWPANs and to examine its impact on network performance. Our method performs key exchange between two parties, but it also handles the ACL table and manages the established secret keys. Our implementation requires approximately 7.7 KBytes of code memory when built with the MSP430 GCC toolchain and used with Tmote Sky devices. This indicates that the implementation is somewhat too large for legacy sensor devices. However, contemporary devices incorporate considerably larger flash storage (e.g. 512KB) and 32-bit MCUs, allowing a significantly extended memory space. This alleviates address space restrictions posed by the original MSP430 and subsequent MSP430X architectures.

Average energy consumption attributed to microcontroller activity for the computation of the public and the secret key (elliptic curve 128-bit scalar multiplication) is less than 48 mJ, and about 96 mJ for the entire key exchange negotiation. Total energy consumption is estimated to be approximately 115 mJ, with the increase being primarily attributed to the radio transceiver in listening and frame reception modes.

By simulating a larger network, we investigated the impact of the key lifetime to the overall energy consumption, network performance and scalability. We observe that packet loss decreases as key lifetime increases. This is due to two factors: i) In our current implementation, lack of a key with a neighbor results in an outgoing packet getting dropped and ii) Key negotiation itself is time-consuming, occasionally leading to further losses. Packet delivery ratio can be improved by pro-actively triggering the establishment of a new key between two neighbors before the existing key times out.

Our scalability investigation reveals that each network node is able to keep up to 36 distinct keys in its ACL table. The approach scales well even in very dense networks; this number is sufficiently large if we keep in mind that each node only needs to establish keys with its single-hop neighbors.

The current solution is unauthenticated and is thus vulnerable to man-in-the-middle attacks, similar to the original Diffie Hellman key exchange. To address this, a method to authenticate the two parties before key negotiation would be required, with the Elliptic Curve DSA (ECDSA) algorithm posing as an attractive candidate. This is left as future work.

In terms of the time required to negotiate a shared secret, the algorithm can be further optimised. As part of our future work, we will adjust the ECDH daemon to generate Public-Private key pairs pro-actively, during node idle periods,

instead of on-demand when a new key request is received. As a result, the only computationally intensive operation that will need to be conducted during key negotiation will be the calculation of the shared secret, which can take place in parallel at the two participating parties. We estimate that this will reduce the total negotiation time by approximately 60%.

In the future, ECC hardware acceleration can be employed to make the approach more viable. Hardware acceleration can have a host of positive effects: i) It can decrease code footprint, since it would mean that the implementation of elliptic curve calculations would no longer need to be included in firmware, ii) Key negotiation will be considerably faster, since ECC calculations will not need to be performed by software, iii) Assuming energy-efficient acceleration hardware, total energy consumption for the negotiation will be lower.

On contemporary sensor devices, encryption of link-layer frames is conducted by AES co-processors and is generally considered to be fast and energy-efficient. However, it is impossible to simulate hardware acceleration within the Cooja simulator. Providing a software AES implementation would have had an obfuscating effect on all metrics under investigation and since this work focused on key negotiation, we deliberately removed layer two encryption functionality in its entirety. As part of our future plans, we will evaluate the method in a real testbed with layer two encryption enabled and performed by hardware. This will allow us to draw conclusions on the viability of the complete solution.

References

1. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next century challenges: Scalable coordination in sensor networks. In: Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, Washington, USA, pp. 263–270 (1999)
2. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGPLAN Not. 35, 93–104 (2000)
3. Dunkels, A.: Full TCP/IP for 8-bit architectures. In: Proceedings of the 1st International Conference on Mobile systems, Applications and Services, New York, pp. 85–98 (2003)
4. Hui, J.W., Culler, D.E.: IP is dead, long live IP for wireless sensor networks. In: Proc. 6th ACM Conference on Embedded Network Sensor Systems (SenSys 2008), New York, NY, USA, pp. 15–28 (2008)
5. Montenegro, G., Kushalnagar, N., Hui, J., Culler, D.: Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944 (2007), `http://tools.ietf.org/html/rfc4944`
6. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
7. Zhang, J., Varadharajan, V.: Wireless sensor network key management survey and taxonomy. J. Netw. Comput. Appl. 33, 63–75 (2010)
8. Roman, R., Alcaraz, C., Lopez, J., Sklavos, N.: Key management systems for sensor networks in the context of the Internet of Things. Computers & Electrical Engineering 37, 147–159 (2011)

9. Winter, T. (ed.), Thubert, P. (ed.), Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J.P., Alexander, R.: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, RFC 6550 (2010), http://tools.ietf.org/html/rfc6550
10. Çamtepe, S.A., Yener, B.: Combinatorial design of key distribution mechanisms for wireless sensor networks. IEEE/ACM Trans. Netw. 15, 346–358 (2007)
11. Bechkit, W., Challal, Y., Bouabdallah, A., Tarokh, V.: A Highly Scalable Key Pre-Distribution Scheme for Wireless Sensor Networks. IEEE Transactions on Wireless Communications 12(2), 948–959 (2013)
12. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 41–47. ACM, New York (2002)
13. Wang, H., Li, Q.: Efficient Implementation of Public Key Cryptosystems on Mote Sensors (Short Paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 519–528. Springer, Heidelberg (2006)
14. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
15. Wander, A.S., Gura, N., Eberle, H., Gupta, V., Shantz, S.C.: Energy analysis of public-key cryptography for wireless sensor networks. In: Third IEEE International Conference on Pervasive Computing and Communications - PerCom 2005, pp. 324–328 (2005)
16. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management part 1: General (revision 3). NIST special publication 800, 57 (2011)
17. Bianchi, G., Capossele, A.T., Mei, A., Petrioli, C.: Flexible key exchange negotiation for wireless sensor networks. In: Proceedings of the 5th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, pp. 55–62. ACM, New York (2010)
18. Mzid, R., Boujelben, M., Youssef, H., Abid, M.: Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography. In: 2010 International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS), Sousse, pp. 1–8 (2010)
19. Sustainable Computing Research (SCoRe) - ContikiECC, http://score.ucsc.lk/projects/contikiecc
20. Liu, A., Ning, P.: TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN 2008, pp. 245–256 (2008)