Key Establishment and Trustful Communication for the Internet of Things

Davi Resner and Antônio Augusto Fröhlich
Software/Hardware Integration Lab, Federal University of Santa Catarina,
PO Box 476, 88040-900 - Florianópolis, SC, Brazil

Keywords: Internet of Things, Trustfulness, Security, Key Establishment.

Abstract: This work describes a practical solution for the problem of cryptographic key establishment and secure communication in the context of the Internet of Things, in which computational efficiency is a fundamental requirement. A symmetric-key establishment protocol based on AES, Poly1305-AES, time synchronization, Elliptic Curve Diffie-Hellman and sensor IDs is proposed to achieve data confidentiality, authentication, integrity and prevention from replay attacks. Such a protocol was implemented in the EPOS operating system in the form of a network layer that transparently provides trustfulness. Tests were executed on the EPOSMoteII platform and the analysis of the results shows that the implementation is adequate to be used in the scenario of embedded systems with low processing power.

1 INTRODUCTION

The idea of an Internet of Things (IoT) is quickly materializing through the adoption of RFID as a replacement for barcode along with the introduction of Near Field Communication (NFC) devices that are able to interface our daily-life objects with the Internet. However, the next steps towards a global network of smart objects will drive us through several large-scale, interdisciplinary challenges. In particular, security and privacy are issues that must be consistently addressed (Atzori et al., 2010) (Elkhodr et al., 2013) before IoT can make its way into people's lives.

Things in IoT will interact with each other and with human beings through a myriad of communication technologies. This communication will often happen wirelessly, and will almost always be subject to interference, corruption, eavesdropping and all kinds of cybernetic attacks. Most of the encryption and authentication techniques developed for the original Internet – the Internet of People – to handle impersonation, tampering, and replay attacks, in theory, can also be applied to the IoT. It is important to recognize, however, that these conventional solutions will not always be adequate for the IoT, making it an open area of research (Elkhodr et al., 2013). The microcontrollers used in smart objects will seldom be able to put up with the requirements. Furthermore, IoT will be subject to particular conditions not so often faced by today's Internet devices. Some *things* will send messages that will trigger immediate reactions from other *things* that directly control the environment. Capturing and reproducing one such valid message, even if it is encrypted and signed, could lead complex systems such as roadways, factories, and even whole smart cities to misbehave. Some *things* will harvest energy from the environment for hours before they can say something to the world, and when they speak, one will have to decide whether or not to believe in what they say without having a chance to further discuss the subject (at least not for a couple of hours). Conventional solutions such as transaction authentication and channel masking (Fu et al., 2003) are of little help in this context.

In previous works, we introduced a trustful infrastructure for the IoT conceived with these pitfalls in mind (Fröhlich et al., 2011; Fröhlich et al., 2013). In this paper, we detail the design and implementation of a key component of that infrastructure: *key establishment*. In that infrastructure, trustfulness is tackled at MAC layer by extensive use of the Advanced Encryption Standard (AES) hardware accelerators now available in most IEEE 802.15.4 implementations. The adoption of this widespread symmetric-key algorithm as the foundation of the infrastructure's trustfulness brings several practical benefits (e.g. performance and power consumption), but brings also the major drawback of symmetric-key encryption: both parties must share a secret key. Generating the keys before deployment and storing them for subsequent distribution would lessen the infrastructure security to the level of key management, involving humans. Generating the keys at servers and sending them to the smart things over the network would also make room for additional attacks, since decrypting the message containing the key at any time would reveal the key itself. Therefore, the proposed key bootstrapping strategy also makes use of Elliptic Curve Diffie-Hellman (ECDH) for distributed key generation, and precise clock synchronization in a communication protocol that exploits previously-known sensor information to overcome such challenges with no need for third-parties.

The remaining of the paper is organized as follows: Section 2 presents the related works. In Section 3, we describe the main components that were used as building blocks for the trustful IoT infrastructure. In Section 4 we expose how we combine the building blocks and propose a key establishment and trustful communication protocol, which is analysed in Section 5. Section 6 concludes the paper.

2 RELATED WORK

TinySec (Karlof et al., 2004) defines a link-layer security architecture for Wireless Sensor Networks (WSNs), providing encryption and authentication. TinySec supports two different security options: authenticated encryption (TinySec-AE) and authentication only (TinySec-Auth). In authenticated encryption mode, TinySec encrypts the data payload according to the Skipjack block cipher and authenticates the packet with a Message Authenticity Code (MAC). The MAC is computed over the encrypted data and the packet header. In authentication only mode, TinySec authenticates the entire packet with a MAC but does not encrypt the data payload. The inclusion of a MAC to ensure authenticity and integrity has a cost on radio usage and consequently in energy consumption, because the hash values commonly represent a long sequence of bits – the length of a MAC determines the security strength of a MAC function (Sun et al., 2010). TinySec achieves low energy consumption by reducing the MAC size, hence decreasing the level of security provided. TinySec also does not attempt to protect against replay attacks and does not discuss how to establish link-layer keys. TinySec is implemented in TinyOS and runs on Mica, Mica2, and Mica2Dot platforms, each one using Atmel processors. TinySec has 3000 lines of nesC code , and the implementation requires 728 bytes of RAM and 7146 bytes of program space.

MiniSec (Luk et al., 2007) is a secure network layer protocol for WSNs that attempts to solve the known problems of TinySec. MiniSec accomplishes this by combining three techniques. First, it employs a block cipher mode of operation that provides both privacy and authenticity in only one pass over the message data. Second, MiniSec sends only a few bits of the Initialization Vector (IV) – a block of bits used by some operating modes to randomize the encryption, producing distinct ciphertexts from the same plaintext over time – while retaining the security of a full-length IV per packet. In order to protect against replay attacks and reduce the radio's energy consumption, it uses synchronized counters, but only sending the last bits of the counter along with each packet. However *Jinwala et al.* showed that such scheme requires the execution of costly resynchronization routines when the counters shared are desynchronized (packets delivery out-of-order) (Jinwala et al., 2009).

Related protocols for key establishment include a fast scheme based on Elliptic Curve Cryptography presented in (Huang et al., 2003). They use the idea of combining symmetric and asymmetric operations, proposing what the authors call a hybrid protocol, which offloads work from the sensors by making the gateway work more. (Pan et al., 2011) shows that some of their security claims do not hold since the gateway could recover a sensor's secret long-term key, and proposes an improved version.

The idea of a sensor-ID based protocol is explored in (Li-ping and Yi, 2009), and a rough estimation of time and bandwidth costs indicates that this can be an efficient approach. In these works, however, either the sensor needs to be pre-loaded with specific and sensitive information or an external security manager is assumed to distribute certificates to sensors through an alternative, secure out-of-band channel. The present work contrasts by aiming at reducing this pre-deployment effort and proposing a practical solution for key establishment.

3 PROTOCOL BUILDING BLOCKS

3.1 EPOS

EPOSMote is an open hardware project (EPOS, 2014). The project's main objective is delivering a hardware platform to allow research on energy harvesting, biointegration, and MEMS-based (Micro-ElectroMechanical Systems) sensors. Figure 1 shows

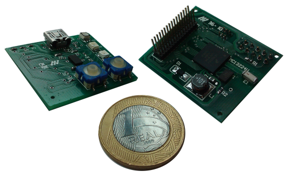
Figure 1: EPOSMoteII SDK side-by-side with a R$1 coin.

EPOSMoteII, the model used in this work. It features a 32-bit, 26MHz ARM7TDMI-S processor, 128kB of flash memory, 96kB of RAM and an IEEE 802.15.4-compliant radio transceiver. EPOS (Embedded Parallel Operating System) is the running operating system of choice, which is implemented in C++.

3.2 PTP

The proposed protocol makes use of timing synchronization to add principles of temporality to the secure communication. In previous work, we implemented EPOS' timing protocol (Oliveira et al., 2012), which delivers clock time across a wireless sensor network in conformance with the IEEE 1588 standard, the Precision Time Protocol (PTP). A central node (the gateway) equipped with a GPS receiver propagates its precise clock time to slave nodes as illustrated by Figure 2. Since in the wireless scenario propagation is done broadcasting messages, listening nodes can passively take advantage of protocol interactions to recalibrate whenever they observe valid PTP messages in the network, greatly reducing the total traffic generated. EPOS' PTP can keep a PAN synchronized with sub-millisecond precision.

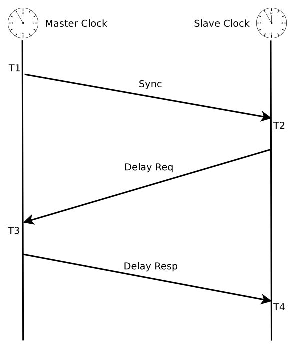

Figure 2: Overview of messages exchanged by EPOS' PTP.

3.3 Hardware AES Acceleration

The Advanced Encryption Standard is a symmetric-key algorithm considered to be resistant against mathematical attacks. It consists of a block cipher containing a 128-bit block size, with key sizes of 128, 192, and 256 bits. AES is a very popular cryptographic engine and is included in the IEEE 802.15.4 standard, so that many devices for the Internet of Things (including EPOSMoteII) have hardware acceleration for AES available. The use of hardware acceleration for cryptographic algorithms not only enhances the performance of security systems but also leaves the computing resources available to a more useful work (Chang et al., 2010). Therefore, we make extensive use of the widespread hardware-assisted security mechanism to derive keys, sign, encrypt and decrypt all necessary data. The implementation in EPOSMoteII uses a fixed key size of 128 bits and is extremely efficient (Fröhlich et al., 2011).

3.4 Poly1305-AES

Poly1305-AES (Bernstein, 2005) is a state-of-the-art Message Authentication Code algorithm that uses AES (thus taking advantage of hardware acceleration) to compute a 16-byte authenticator according to Equation 1, where:

- c is of length q, and is derived from the message,
- k, r are two secret keys, each of length 16,
- n is a *nonce*,
- $AES_k(m)$ is the ciphertext produced by running AES using message m and key k as input.

$$Poly(c,k,r,n) = (((c_1 \cdot r^q + \cdots + c_q \cdot r^1) \bmod (2^{130}-5)) + AES_k(n)) \bmod 2^{128} \quad (1)$$

Poly1305-AES is proven to be secure if AES is secure (Bernstein, 2005), and can be implemented with comparative efficiency.

3.5 Diffie-Hellman

Elliptic Curve Diffie-Hellman (ECDH) over Prime Curves (F_p), widely considered more robust and efficient than the traditional Diffie-Hellman scheme (NSA, 2009), is used in the present work. The protocol, however, is agnostic as to which underlying arithmetics for Diffie-Hellman is used.

4 SECURE KEY BOOTSTRAPPING PROTOCOL

The proposed protocol adds two-way authentication to a typical Diffie-Hellman scheme, and also allows an additional secret to be shared between a sensor and the gateway: a deployment time window. It relies on unique sensor IDs, AES, Poly1305-AES and synchronized clocks. The following conditions are assumed to be valid by the protocol. Some of these are discussed further in Section 5.

- Each sensor has a unique identifier on the network. This identifier needs not have a specific format. It can be, for instance, the microprocessor's serial number.
- Sensors are capable enough to perform assymetric cryptography operations when they join the network.
- There is a node called *gateway* which is a local, trusted and controlled machine.
- Secure communication is only necessary between sensor and gateway. In WSNs, usually, most (or all) communication happens between sensors and gateway only.
- Sensors have persistent memory for holding a cryptographic key; i.e. sensors are assumed not to "forget" keys (e.g. due to a power loss) once they are established.

The first three conditions are inherent to the protocol, while the others are stated to limit the scope of the present work.

The following notation is used through the rest of this paper:

- ID_a: Unique identifier of sensor a.
- $Auth_a$: A hash computed from ID_a, for instance $AES(ID_a, ID_a)$.
- (Y_a, P_a): Public and private (respectively) Diffie-Hellman key-pair of sensor a.
- (Y_g, P_g): Gateway's DH key-pair.
- $DH(P_a, Y_b)$: a Diffie-Hellman operation, which calculates a Master Secret K_{ab}. In our implementation, this denotes an Elliptic Curve point multiplication by a scalar.
- $Poly(a, b, c, d)$: The result of running the Poly1305-AES algorithm using a, b, c, d as parameters (Equation 1).
- $M_1 \cdot M_2$: Concatenation of messages M_1 and M_2
- $\{M\}_k$: Message M encrypted under symmetric key k, e.g.: $AES(M, k)$.
- T: The current time window of the network, which is the timestamp truncated to a fixed, network-wide interval length. The windows shall be long enough to accommodate a message's transmission and reception.

The protocol is divided into 3 phases: Initialization, Key Establishment and Authentication, after

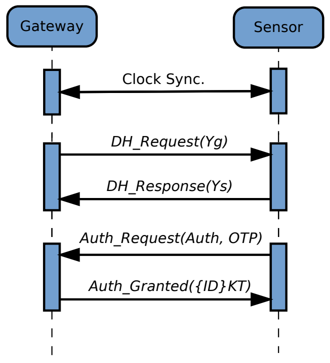

Figure 3: Overview of messages exchanged for key establishment and authentication.

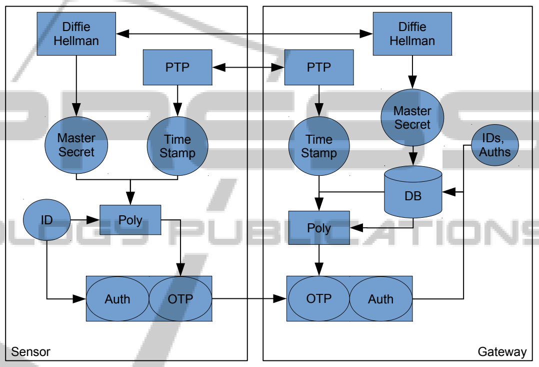

Figure 4: Overview of the interactions between several of the protocol's building-blocks.

which nodes can start communicating trustfully. Figure 3 illustrates the messages exchanged and Figure 4 presents an overview of the interactions between the various building-blocks, as detailed in the following sections.

4.1 Initialization

Each trusted sensor is assigned a unique identifier. The ID need not have any particular form, as long as it is unique in the network. It can be the device's serial number, for instance. The IDs of each sensor to be trusted must be loaded in the gateway, and is delegated to the user to do this in a secure way, since the gateway is assumed to be a trusted, controlled local machine. Sensors also have an *Auth* code, which is a known hash function of the ID, and the gateway has a database corresponding *Auth* codes with IDs.

When a sensor is deployed, it listens to the Diffie-Hellman parameters of the network (which may optionally be pre-loaded, since they are public and usually known before deployment), generates its Diffie-Hellman key-pair, synchronizes its clock with the network and remains idle until the gateway asks for authentication. Once the user loads the IDs in the gateway, it is ready to issue a $DH\_Request(Y_g)$ command, moving on to the next step.

4.2 Key Establishment

Key establishment is carried out as a traditional Diffie-Hellman protocol. The first message is sent by the gateway and is denoted *DH_Request(Y_g)*. The part in parenthesis indicates a variable value sent along with the message (in this case, Y_g). This message can be broadcast or transmitted to a particular node. When a *DH_Request(Y_g)* message arrives at the destination, the receiving sensor s must respond with *DH_Response(Y_s)* and then calculate the symmetric key $K_{sg} = DH(P_s, Y_g)$, which we call *Master Secret*. Similarly, upon receiving the response the gateway calculates $K_{sg} = DH(P_g, Y_s)$. At this point, sensor and gateway share a symmetric key. However, since no authentication aspects have been involved so far, they have no evidence that this key is shared with a trustful node in the network.

4.3 Authentication

The authentication step's purpose is twofold: it provides evidence to the gateway that the sensor is authentic (i.e. holds a valid and unique ID) and vice-versa, then ties the previously calculated K_{sg} to this valid ID, validating the Master Secret. In order to do this, the sensor calculates a One-Time Password OTP_s as in Equation 2 and sends to the gateway the message *Auth_Request($Auth_s, OTP_s$)*.

$$OTP_s = Poly(K_{sg} \oplus ID_s, K_{sg}, ID_s, T) \qquad (2)$$

Upon receiving an expected *Auth_Request($Auth_s, OTP_s$)* message, the gateway verifies if there is a Master Secret pending authentication which is implicitly contained in OTP_s. The process starts with a database query for ID_s via $Auth_s$. Then, for each K_{ag} pending authentication, the gateway calculates $OTP_a = Poly(K_{ag} \oplus ID_s, K_{ag}, ID_s, T)$ (using for every calculation the T indicating the moment the *Auth_Request* message was received) and checks if $OTP_a = OTP_s$. If and only if OTP_s is found, the gateway has evidence that K_{ag} is in fact K_{sg} and was shared with an authentic sensor identified by ID_s; it then associates the ID with the Master Secret for future communication with s.

Finally, the gateway transmits the last message: *Auth_Granted($\{ID_s\}_{KT_{sg}}$)*, to provide to the sensor evidence that this node is an authentic gateway (i.e. is the only other node that knows ID_s). KT_{sg} is a disposable key derived from the Master Secret with the mechanism explained in Section 4.4.

Upon receiving *Auth_Granted($\{ID_s\}_{KT_{sg}}$)*, the sensor is able to derive KT_{sg} as well and decrypt the message to check if ID_s is correct. If it is, the sensor starts trusting the Master Secret. Otherwise, it shall not consider itself authenticated and must start the protocol again, waiting for a new *DH_Request* message.

4.4 Secure Communication

Once a Master Secret K_{sg} has been authenticated, it is ready to be used as an AES key, but only indirectly. When a node needs to send a message M securely, it must derive a key KT_{sg} according to Equation 3, include a checksum C to the message and then send *Secure_Message($\{C \cdot M\}_{KT_{sg}}$)*.

$$KT_{sg} = Poly(K_{sg} \oplus ID_s, K_{sg}, ID_s, T) \qquad (3)$$

If the receiving node is a sensor, it is able to derive KT_{sg}, decrypt the message to obtain C and M and check if the checksum is correct (if it isn't, the message is discarded). If the receiver is the gateway, it must derive a KT_{ag} for each authenticated sensor a, then try to decrypt the message with each key until one generates the correct checksum (if none generates, the message is discarded). If the implementation allows access to a lower-layer network address, it can be used as a heuristic to accelerate this search for K_{sg}.

When C is correctly generated, the receiver has evidence that the *Secure_Message* was generated by the only other node in the network that knows the previously validated Master Secret. It then trusts that the message is authentic, confidential, wasn't modified and was generated within the current network time window.

5 EVALUATION

This section aims at evaluating the proposed protocol's efficiency and viability. The analysis is divided in two: Section 5.1 brings a qualitative discussion regarding the protocol's security aspects, its main positive characteristics and limitations. The section that follows it, 5.2, shows results of tests performed with a prototype implementation of the protocol.

5.1 Protocol Analysis

IoT devices will often communicate through wireless technology, allowing any radio interface configured at the same frequency band to monitor or participate in communications, which is very convenient for attackers (Zhou et al., 2008). In order to avoid undesired attacks a secure infrastructure for the IoT must provide the principles of *confidentiality*, *authenticity* and *integrity* (Suo et al., 2012).

In the proposed protocol, if a secure message $M' = \{C \cdot M\}_{KT_{sg}}$ is sent – for example – from s to g and is successfully decrypted yielding a matching checksum, the receiver can conclude that:

- M is **confidential**, since only the sensor and the gateway know K_{sg}.
- M is **authentic**, because K_{sg} is secret and tied to ID_s.
- M' was **not modified** by anyone but s, because it is extremely unlikely that the message was altered blindly and still carries a valid checksum.
- M' was generated in the current network time window T.

5.1.1 IDs

Each sensor's unique identifier does not need to have a particular format, but some observations impact the provided security level. An attacker can disguise itself as an authentic sensor if it finds a valid ID and uses it in the appropriate time, before the legitimate holder of that ID does (see Section 5.1.3); therefore, it is desirable for any valid ID to be very hard to find. The three main factors that impact on this difficulty is the secrecy with which the IDs are stored and handled, their size in bits and their predictability.

Secrecy is a factor we can't control, since it greatly depends on the context in which the protocol is being used, including human and logistic factors such as the outsourcing of the team that generates and instals the sensors and their IDs. Ideally, the IDs should be kept secret and only revealed in a controlled way to those who necessarily need to know them.

ID size is directly related to the security level. The larger the ID is, the harder it will be for an attacker to blindly guess it correctly. The *Auth* codes and the One-Time Passwords also benefit from an increase in ID size if an AES encryption mode that takes advantage of the whole plaintext is used (e.g. CBC mode). An ID smaller than 128 bits is highly unrecommended, because it would affect the values utilized as keys in *Auth* and OTP, which would decrease AES security.

Predictability is an important aspect because an attacker that has an idea of the format of valid IDs can potentially find one much more easily by trial-and-error. Thus, randomly-generated IDs are the most recommended.

In summary, ideally one would like the IDs to be random, 128-bit (or larger) numbers, which are revealed only to trusted individuals who necessarily need to know them. One should balance these three factors as the context allows. If, for instance, one wishes to use each device's not-secret, 10-bit serial number as IDs, one could instead use the serial numbers as input to a random number generator of greater magnitude and use the output as ID.

With these recommendations in mind, it may seem easier to simply load pre-established symmetric keys on the sensor, or use the IDs directly as keys and dismiss any protocol. In this strategy, however, if an attacker happens to find an ID (or key, in this case) they would be able to decrypt *all* the messages ever exchanged – in the past or in the future – by the corresponding node, while in our proposal the attacker must, besides finding out an ID, break the discrete logarithm problem for elliptic curves, which in itself is far from a trivial problem. One could also use the shared, secret IDs as Diffie-Hellman keys and dismiss the rest of the protocol, but that would bring the known Diffie-Hellman Man-In-The-Middle vulnerability (since the sensor would not be able to recognize a valid gateway) and wouldn't provide temporality of messages.

Because the proposed protocol does not derive keys exclusively from the ID, security holds even if they are employed with low care (see Section 5.1.3), allowing for some relaxing in the requirements regarding the ID. Using the IDs directly as keys would require a care directly proportional to the desired security level.

5.1.2 Timestamps

The timestamps used along the protocol are not the direct value of the clock, but a truncation to a pre-established, network-level time window, which will define the timestamp's granularity. There should be here a point of contact with the lower layers of the network stack since the timestamp used in the calculations should reflect the moment in which the message is actually transmitted. In WSNs, for instance, the use of duty-cycling protocols is common, in which senders must transmit preamble packets for X units of time before sending the actual message. In this case, the timestamp used for encryption on that message should be T_{i+X} (where i is the current moment).

Within a single timestamp, a message must be able to travel from the sender to the receiver. Timestamps with too coarse granularity should not be used because replay attacks are possible within a same timestamp.

5.1.3 Possible Attacks

In this section we consider a few possible attacks to the protocol, their potential impacts and preventive measures.

1. *Guessing an ID*
   Attack. The attacker deploys an unauthorized device which will try to authenticate using a number of different educated (or completely in-the-dark) guesses of ID, until it finds a valid one.
   If Successful. An unauthorized node is deployed and treated as a trusted one. If the legitimate holder of the ID tries to authenticate later, it will be treated as an attacker and blocked (see next attack).
   Countermeasure. Since the gateway is the one to start the protocol, it has knowledge of how many authentication attempts should be happening at any given time. If too many attempts are detected, the gateway can start ignoring them, so an attacker cannot try an arbitrary number of IDs. Furthermore, any node that is detected as trying to authenticate more than a very few times could be blacklisted and ignored. A mechanism for trusted nodes to detect and report such malicious nodes is being devised as future work (it could take geographic location information, for instance). If the very first guess is correct, this attack turns into the next one.

2. *Knowing an ID*
   Attack. The attacker knows a valid ID and deploys a malicious node using it.
   If Successful. An unauthorized node is deployed and treated as a trusted one. If the legitimate holder of the ID tries to authenticate later, it will be treated as an attacker and blocked.
   Countermeasure. The gateway can determine if there is already some trusted node under that ID. If a different node is trying to use the ID while some node is already using it, the gateway blocks the one that came later. This attack is only successful if the attacker finds out the valid ID, deploys and authenticates the node before the proper node is authenticated; therefore the attacker must not only find out the ID by some means, but also use it with perfect timing.
   This solution implies that if a legitimate node is reset for some reason and "forgets" its key without the gateway knowing, it will not be able to authenticate again. We assume that this won't happen, but if it is a concern, a policy of key renewing can be implemented, which periodically invalidates keys and ensures that the key bootstrapping protocol is run to establish new, valid keys. This way a reset sensor will be inactive only until the next key renewal period, but, on the other hand, every such period will open a new window for successful "ID-knowing" attacks.
   Since the generation of a shared key is entirely independent of the ID, stealing an ID after the key is established does not compromise the key in any way.

3. *Manipulating Time Synchronization*
   Attack. Since time synchronization is needed for secure communication, PTP must be run unsecured by the protocol at least for the first time, enabling attackers to manipulate PTP messages and break synchronization.
   If Successful. New keys will not be established successfully and secured messages will not be decrypted correctly, resulting in Denial of Service (DoS).
   Countermeasure. DoS is a threat to wireless networks in general, since attackers could in most cases simply *jam* the channel by inserting a large number of messages, preventing *any* communication from being successful. Since this is the only harm caused by a PTP attack (no cryptographic material can be obtained this way), we do not regard this as a new vulnerability.

4. *Node Capture*
   Attack. The attacker physically acquires a legitimate node in the network.
   If Successful. The attacker is assumed to have complete control over the node, and can read and modify its data as they like.
   Countermeasure. This attack is also a general threat to Wireless Sensor Networks. By cloning all the node's already validated cryptographic information to a new one, the attacker can effectively introduce a malicious node in the system, which will be treated as a trusted one. Nevertheless, since any given node holds no information – explicit or implicit – about any other node's key, it is guaranteed that capturing a node will not disclose private cryptographic-key data about the gateway (besides one shared key) or any other nodes in the network.

5.2 Implementation Analysis

This section analyses the performance of the code written to implement the protocol in EPOS (EPOS, 2014), allowing an application to communicate trustfully with transparency. In the test scenario, an EPOSMoteII device (Section 3.1) was used as the gateway and two more as sensors. The implementation leaves room for optimization (e.g. more complex algorithms for elliptic curve operations, hardware implementation ...), and the aim of the evaluation is to show that the protocol implies arguably modest overhead, even with a sub-optimal implementation.

We implemented the Elliptic Curve arithmetic – following the works of (Brown et al., 2001; Menezes et al., 1996) – and Poly1305-AES libraries completely in C++. Table 1 shows the resulting code sizes for each part.

In the running time graphs, each bar represents the mathematical average of a few iterations of the indicated algorithms and parameters, running on different EPOSMotes. Network overhead is not accounted. The Elliptic Curves used are the ones recommended in (SEC, 2000), and the numbers in the names (e.g. secp128r1) indicate the size in bits of the parameters.

Table 1: Size (in bytes) of .text section for, respectively, the Elliptic Curve Cryptography, Big Integer and Poly1305-AES libraries.

	ECC	Bignum	Poly1305	Total
size	2880	3700	1512	8092

5.2.1 Poly1305-AES

Figure 5 shows the time taken by our implementation of the Poly1305-AES algorithm for derivation of *KT* keys (which must be performed every time a message is encrypted) and One-Time Passwords, according to Equation 2. The total time taken by the gateway for checking an OTP and preparing a confirmation message is shown in Figure 6.

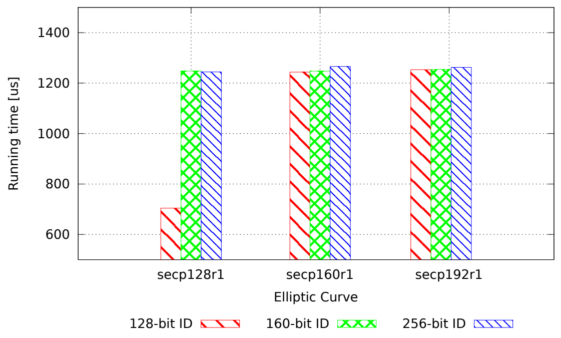

Figure 5: Average times (μs) for OTP derivation.

In both graphs, there is a significant difference in the case where the ID and the ECDH parameters are 128-bit wide. This difference happens because

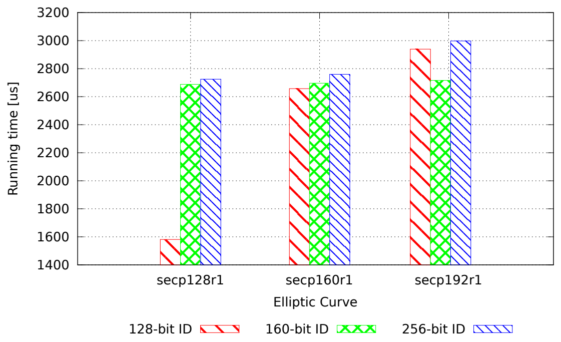

Figure 6: Average times (μs) for authentication checking and confirmation.

Poly1305-AES operates on 128-bit blocks, and in every other case at least one of the parameters is larger than this.

5.2.2 Elliptic Curve Diffie-Hellman

The execution times shown in the two previous figures are insignificant compared to the time taken by the ECDH implementation. Figure 7 shows the average times taken to perform *one* of the two Elliptic Curve point multiplications required by the protocol. One should keep in mind, however, that Poly1305-AES is used whenever a secure message is sent and received, while this multiplication pair generates a key that will be valid for a very long time (hours, days, months, depending on the application), therefore they will happen very sporadically, in some cases only once in a sensor's lifetime in the network.

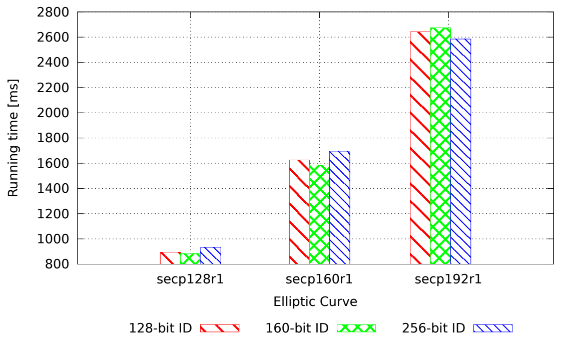

Figure 7: Average times (ms) for ECDH point multiplication.

5.2.3 Communication Overhead

For key establishment, a total of 4 messages are exchanged, as shown in Figure 3. For each message, the security layer implementation uses one byte to represent message type, and a payload of size specific for each message. The first two messages each carry a Diffie-Hellman public key. Its size is implementation-dependent, and in our ECDH implementation it is a two-dimensional Elliptic Curve point. The total size of the point is equal to twice the size of the generated Master Secret. In $Auth\_Request(Auth, OTP)$, both values are fixed at 16 bytes each. $Auth\_Granted(\{ID_s\}_{KT_{sg}})$ has a payload of size equal to ID_s. Table 2 shows the total communication overhead for these 4 messages, for varying sizes of ID and Master Secret.

Table 2: Total communication overhead (in bytes) for key establishment for varying sizes of Master Secret (rows) and ID (columns).

ID size / **MS size**	**16**	**20**	**32**
16	116	120	132
20	132	136	148
24	148	152	164

6 CONCLUSIONS

This paper presented a trustful infrastructure for the IoT developed within the realm of project EPOS. Aspects such as people privacy in respect to traffic pattern analysis and data dependability have not been considered in this paper. Also, optimized implementations and secure group communication are topics left as future work and are currently under study.

The proposed infrastructure was implemented around the EPOSMoteII platform and delivered to end users through a trustful communication protocol stack. Trustfulness for the infrastructure was achieved through a combination of mechanisms. A practical key establishment protocol based on AES, Poly1305-AES, time synchronization, Diffie-Hellman and sensor IDs was proposed to achieve confidentiality, authentication, integrity and prevention from replay attacks. The proposal was experimentally evaluated in terms of running time in a real-world implementation. The results confirm that the proposed infrastructure can provide the security needed without introducing excessive overhead to a network of things, a key step in making the Internet of Things a daily reality.

REFERENCES

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805.

Bernstein, D. J. (2005). The poly1305-aes message-authentication code. In *Proceedings of Fast Software Encryption*, pages 32–49, Paris, France.

Brown, M., Hankerson, D., López, J., and Menezes, A. (2001). Software implementation of the nist elliptic curves over prime fields. In Naccache, D., editor, *Topics in Cryptology - CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265. Springer Berlin Heidelberg.

Chang, J.-T., Liu, S., Gaudiot, J., and Liu, C. (2010). Hardware-assisted security mechanism: The acceleration of cryptographic operations with low hardware cost. In *Performance Computing and Communications Conference (IPCCC), 2010 IEEE 29th International*, pages 327 –328.

Elkhodr, M., Shahrestani, S., and Cheung, H. (2013). The internet of things: Visions & challenges. In *TENCON Spring Conference*, pages 218 – 222.

EPOS (2014). Epos project website. http://epos.lisha.ufsc.br.

Fröhlich, A. A., Okazaki, A. M., Steiner, R. V., Oliveira, P., and Martina, J. E. (2013). A cross-layer approach to trustfulness in the internet of things. In *9th Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, Paderborn, Germany.

Fröhlich, A. A., Steiner, R., and Rufino, L. M. (2011). A trustful infrastructure for the internet of things based on eposmote. In *9th IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 63–68, Sydney, Australia.

Fu, X., Graham, B., Bettati, R., and Zhao, W. (2003). Active traffic analysis attacks and countermeasures. In *Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing*, ICCNMC '03, pages 31–, Washington, DC, USA. IEEE Computer Society.

Huang, Q., Cukier, J., Kobayashi, H., Liu, B., and Zhang, J. (2003). Fast authenticated key establishment protocols for self-organizing sensor networks. In *Proceedings of the 2Nd ACM International Conference on Wireless Sensor Networks and Applications*, WSNA '03, pages 141–150, New York, NY, USA. ACM.

Jinwala, D., Patel, D., Patel, S., and Dasgupta, K. (2009). Replay protection at the link layer security in wireless sensor networks. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 1, pages 160 –165.

Karlof, C., Sastry, N., and Wagner, D. (2004). Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 162–175, New York, NY, USA. ACM.

Li-ping, Z. and Yi, W. (2009). An id-based key agreement protocol for wireless sensor networks. In *1st International Conference on Information Science and Engineering (ICISE)*, pages 2542 – 2545.

Luk, M., Mezzour, G., Perrig, A., and Gligor, V. (2007). Minisec: A secure sensor network communication architecture. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 479 –488.

Menezes, A., van Oorschot, P., and Vanstone, S. (1996). *Handbook of Applied Cryptography*. CRC Press.

NSA (2009). The case for elliptic curve cryptography. www.nsa.gov/business/programs/elliptic_curve.shtml.

Oliveira, P., Okazaki, A. M., and Fröhlich, A. A. (2012). Sincroniza cão de tempo a nível de so utilizando o protocolo ieee1588. In *Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, Natal, Brazil.

Pan, J., Wang, L., and Ma, C. (2011). Analysis and improvement of an authenticated key exchange protocol. In Bao, F. and Weng, J., editors, *Information Security Practice and Experience*, volume 6672 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg.

SEC (2000). *Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters*. Certicom Research.

Sun, H.-M., Chang, S.-Y., Tello, A., and Chen, Y.-H. (2010). An authentication scheme balancing authenticity and transmission for wireless sensor networks. In *Computer Symposium (ICS), 2010 International*, pages 222 –227.

Suo, H., Wan, J., Zou, C., and Liu, J. (2012). Security in the internet of things: A review. In *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, volume 3, pages 648–651.

Zhou, Y., Fang, Y., and Zhang, Y. (2008). Securing wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 10:6–28.