

Delayed Key Exchange for Constrained Smart Devices

Joona Kannisto¹(✉), Seppo Heikkinen¹, Kristian Slavov², and Jarmo Harju¹

¹ Tampere University of Technology, Tampere, Finland
{joona.kannisto, seppo.heikkinen, jarmo.harju}@tut.fi
² Ericsson Research, Jorvas, Finland
kristian.slavov@ericsson.com

Abstract. In the Internet of Things some nodes, especially sensors, can be constrained and sleepy, i.e., they spend extended periods of time in an inaccessible sleep state. Therefore, the services they offer may have to be accessed through gateways. Typically this requires that the gateway is trusted to store and transmit the data. However, if the gateway cannot be trusted, the data needs to be protected end-to-end. One way of achieving end-to-end security is to perform a key exchange, and secure the subsequent messages using the derived shared secrets. However, when the constrained nodes are sleepy this key exchange may have to be done in a delayed fashion. We present a novel way of utilizing the gateway in key exchange, without the possibility of it influencing or compromising the exchanged keys. The paper investigates the applicability of existing protocols for this purpose. Furthermore, due to a possible need for protocol translations, application layer use of the exchanged keys is examined.

1 Introduction

The Internet of Things (IoT) concept means that each and every device or thing can be accessed through network [1]. This enables new services and opportunities, but also generates a number of security requirements and needs for security related services. For instance, the amount of network connected devices in IoT can range depending on the use case, from as low as tens of nodes to even thousands of nodes, and the deployment environments and scenarios for IoT are versatile. Common examples include industrial systems, home automation, and intelligent traffic related services. Accordingly, the deployments can have justifications based on cost savings, safety, and even entertainment. Therefore, to make the deployment feasible for even the low margin scenarios, the nodes are often envisioned to be energy efficient and low-cost. Because of this requirement of energy efficiency and low cost, some of these devices are constrained and must spend considerable time in inactive sleep state. For example, 8-bit micro-controllers, which would be awake only for a few seconds each minute, could be considered as baseline examples of such devices.

However, end-users, as well as applications in networked devices often require that the services are available on-demand and that they respond with reasonable

delay. With sleeping constrained nodes this can be accomplished by utilizing a small number of more resourceful gateway devices that support the constrained nodes and make the services accessible to the end-users and other devices. This can mean, for example, protocol translations for the more capable end-user devices, so that they can access services in a standard manner.

While gateways enable energy efficiency for the smaller nodes, the use of a gateway can be problematic whenever the gateway is not trusted by both communication parties. Indeed, we might not trust the gateway in all aspects, and the integrity and confidentiality of the messages may have to be preserved end to end by using cryptography.

New protocols have been designed that suite constrained devices better, such as, Constrained Application Protocol (CoAP). Yet, since some of the clients using the services might support only legacy protocols, the gateway may have to do protocol translations as well [2,3]. However, it is not possible to do protocol translations if tunneling protocols are used [2]. Therefore, it is assumed that end-to-end security is handled at the application layer data objects, we explore how the keys from traditional key exchanges could be utilized in securing the data objects.

The following list summarizes the requirements and the problem statement for the presented IoT key exchange scenario:

- Constrained devices are sleeping most of the time, and are not available for direct communication without the help of a gateway device.
- The gateway may not be in the same trust domain, and trusting it should not be necessary.
- The gateway may need to do protocol translations from HTTP to CoAP.
- Cross-layer approaches offer advantages for constrained devices.

In the paper we evaluate existing key exchange protocols, and how they could be adapted to better fit into these requirements.

2 Related Work

The Internet of Things vision is for the Internet to expand into our every day objects. This requires some degree of interoperability with legacy solutions. Indeed, deployment of new protocols has traditionally been slow, and new solutions tend to be built on top of existing ones. Perhaps the most common key exchange protocols within the Internet context are Internet Key Exchange (IKE) [4] and Transport Layer Security (TLS) [5], even though the latter provides a more comprehensive framework for protecting the actual communication as well. Datagram TLS is an adaptation of TLS which makes it more suitable for constrained devices.

Constrained Application Protocol (CoAP) [6] is envisaged to be the way of providing RESTful, i.e. HTTP compatible services, within IoT. CoAP uses User Datagram Protocol (UDP) for transport instead of TCP. Therefore, Datagram TLS (DTLS) [7,8], which is a modification of TLS, is recommended for use with

CoAP [6]. Also IPsec, which uses IKEv2 [4] is mentioned as an option. End-users might not be able to support CoAP, for example, due to firewall policies and lack of application support. Therefore, protocol translations from HTTP to CoAP would need to be done by an intermediary device [3]. This poses a challenge for proposals, where a tunneling protocol, such as IPsec or DTLS is used to provide end-to-end security below CoAP [2, p. 18].

Key exchange could be done at the application layer as well. For example, DTLS handshake could be transmitted as application layer messages [9, p. 14]. In addition, WS-Trust [10] specifies an application layer key exchange, yet, it may not be suitable for constrained devices due the verbosity of the messages [11]. More compact format JSON Web Encryption (JWE) [12] defines public key and symmetric key based key distribution mechanisms for application layer, which are used by Sethi et al. [13] for constrained devices. However, the JWE specification excludes key exchange [12, p. 20].

Bianchi et. al. [14] suggests a key exchange protocol for Wireless Sensor Networks (WSN) that is loosely based on TLS. This protocol tries to minimize the transfer overhead but does not utilize the infrastructure. In addition, modifications to DTLS for constrained devices have been suggested [9], but infrastructure support for key exchange is a less investigated topic.

Proxy based key exchanges, such as Needham–Schroeder protocol [15] are well established, yet we assume a situation where the proxy is not trusted. In addition, using a gateway to mirror sensor readings is a well established practice in the IoT-world [16]. However, the security in these proposals is either not end-to-end, or could be based on key distribution rather than key exchange [13]. Indeed, we consider key exchanges, where both parties contribute to the resulting key material. Proposals where a middle man acts in a supportive role without a direct contribution to the resulting key material are more in this paper’s scope [17, 18]. Yet, we aim to accomplish this in a way that is transparent to the initiator of the exchange.

3 Overview of the Delayed Key Exchange Proposal

3.1 Assumptions and Security Properties

We base this proposal on a scenario where there is a *user*, a *gateway*, and a constrained device, which we will refer to as a *sensor* later on in this document. The user initiates the key exchange with the sensor. We expect that the user conforms to existing protocol specifications. The user’s intention might be, for example, to fetch periodic readings of the sensor data. The gateway, which could also be called a reverse proxy, acts as a middle man between the sensor and the user. Its role is to enable the key exchange in an efficient way overcoming the sensor’s constraints. There might be constraints in the access network of the sensor as well, such as narrow bandwidth or long delays. Also, the same gateway may serve multiple possibly unrelated sensors, each with independent security associations.

While the control of the participating nodes and the infrastructure might be distributed, some expectations can be made about the behavior of the nodes.

For example, the sensor could rely on the gateway to transmit its messages reliably, while the user should instead retransmit its requests as needed. In addition, neither one needs to trust the gateway to store or process the message contents unencrypted. The gateway assumes that the sensor and the user behave correctly but does not need to rely on that in any critical way.

We consider the initiating party i.e. the user as the prime actor, and focus mainly on authenticating the responder. Authenticated key exchange protocols usually support mutual authentication. For example, Pre Shared Key (PSK) based key exchanges authenticate both parties quite naturally. Moreover, they are usually the best choice for constrained environments whenever there is a possibility to setup symmetric secrets. Public key based mutual authentication is supported by the key exchange protocols, as well. However, end-users do not usually have verifiable public keys, and processing certificate chains can be a difficult task for a constrained device. Therefore, it is expected that application layer authentication credentials are used. Additionally, a viable approach to access control is to authenticate the user and decide on the access control policy by a more resourceful third party.

We base the used threat model on the Dolev-Yao -model [19]. Therefore, we expect the attacker to be able to read and insert messages to the channel but not break any strong cryptographic functions.

3.2 Pre-configuration and Delegation

As the sensor is sleeping most of the time, it is expected that the sensor uses the gateway to act as its representative and fetches information about requests periodically from there. This requires a pre-configuration phase, which means agreement on the messages that can be cached by the gateway. Pre-configuration steps differ from general service discovery scenarios by the inclusion of the gateway. Basically, the user will have to know that certain gateway is responsible for a resource that would normally be tied only to, for example, sensor's identity or locator. Moreover, it may be possible that the users will access the services without knowledge about the actual node providing the needed resource [20].

Authentication for the configuration between the gateway and the sensor could be handled either manually or in an opportunistic fashion. Opportunistic measures can be desirable, when better scalability is desired. In other words, in the initial negotiation the parties would learn their identities, which they would later trust. Such crypto-identities are also suggested in CoAP security architecture Internet draft [21].

If the gateway is not acting as a completely transparent device, the sensor should also provide authorisation, which will show that the gateway is authorised to act on its behalf. Moreover, there might even be need to signal more complex trust relationships. In other words, the sensor may have to have means to specify the actions that the gateway can be trusted to do on the sensor's behalf. Other pre-configurable items can include puzzle, or similar mechanisms for ensuring that the user is "honest" in his attempt to communicate. There might also be a

need to share information needed for nonces, which can be used against replay attacks, especially if the gateway cannot be trusted to generate them.

3.3 Delayed Key Exchange

We assume a situation where the sensor wakes up on timed intervals, or as a result of some event. After waking up it sends any pending messages, and fetches new incoming messages from its designated gateway, processes those messages, sends the necessary replies, and goes back to sleep. Indeed, the sensor may rely on the gateway being available, and even delay going to sleep in order to receive its response. Yet, it should not delay sleep waiting for external resources or clients.

The flow of events on a general level is depicted in Fig. 1. The user sends an initialisation message, which indicates his willingness to establish a secure communication to a service in question. Typically the gateway will issue some sort of a challenge, which is basically used to mitigate Denial of Service (DoS) concerns. Note that these parameters are not secret per se, i.e., the adversary will be able to learn them anyway and trying to change them basically results in DoS condition, which could be done anyway if the attacker is controlling the channel.

The user responds to the challenge (solve puzzle, re-send cookie, etc.) and sends the next message (*Response* in Fig. 1). Depending on the use case, the user might also need some authorisation statements to show that he really is allowed to access the service in question (i.e., gateway performs access control). Note that instead of sending this type of authorisation in the first message, it is better to do it here, because this way the gateway does not need to engage in any heavy processing after the first message. Access control could be based on user identities and predefined policies as well.

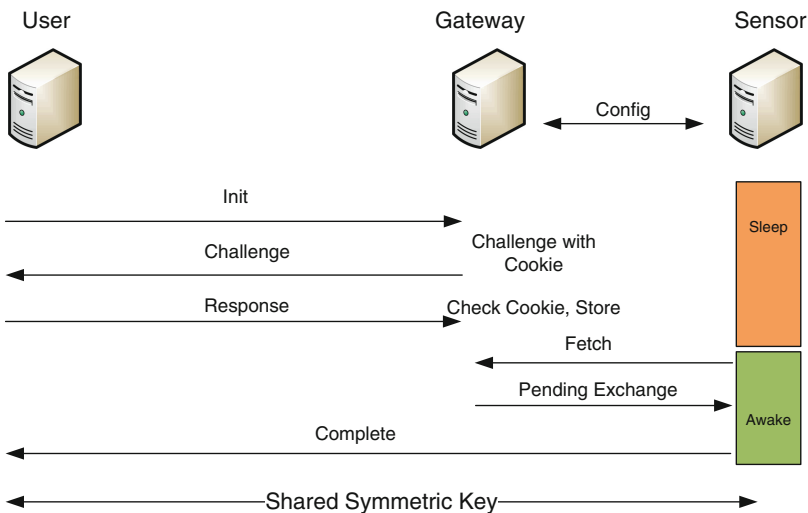


Fig. 1. Delayed key exchange on a general level

From the performance perspective, it is better if also the gateway is able to check any nonces or puzzles, so one needs to ensure that the gateway is in possession of proper information to do this. This might be a requirement if the gateway is responsible for storing the received valid messages, so that they can be later fetched by the sensor.

In case of multiple simultaneous requests for key exchange, the gateway has to implement suitable queuing, and possibly prioritisation, mechanisms. This may also protect the sensor from Denial of Service (DoS) attacks [22]. Depending on the application, it may be necessary for the gateway to send some sort of provisional acknowledgement to the user, so that the user can assume that the message has been received. Yet, the user may choose not to rely on these in a critical way, and timeout arrangements at the user side might be needed anyway.

In the fetching phase the sensor wakes up at its designated time and is then responsible for acquiring the legitimate pending handshake messages, which are waiting to be processed. It is worth noting that the fetching of messages may require the sensor to be awake a longer period of time as it needs to wait for the response to its fetch request.

Once the sensor has processed the messages, it will use the key exchange protocol's key agreement mechanism to generate the common keying material, based on the information it has received. After that it sends its response to the user, which will include sensor's contribution to the keying material if the gateway has not been able to transmit it in earlier messages. After this message, the user is also able to create the same keying material and establish the necessary common secrets. Once the common secret has been created, the user can transmit application level requests, for example, by using HTTP, and can protect the data object within. Also, the gateway may translate the HTTP to CoAP, but it needs to retain the data objects as they are.

3.4 Utilizing the Keys to Protect Application Layer Data

As the sleeping sensor is likely to utilize protocols that do not require sessions, for example CoAP, the gateway may have to be responsible for protocol translations. Therefore, one of the most interesting usages for the key material from the delayed exchange is securing data objects. An example approach presented here uses JavaScript Object Notation (JSON) to represent the data objects, which are to be secured. Signatures and encryption in JSON context are considered in JSON Web Signature [23], and JSON Web Encryption [12] draft specifications and we borrow concepts from there.

Listing 2.1. Protected data object header, comments and linebreaks for clarity

```
{
  "typ": "JWT",
  "alg": "HS256",
  //SPI or TLS session identifier
  "kid": 12010000001,
  "jti": "132312312" // unique identifier
}
```

The Listing 2.1 illustrates what a protected data object header might look like. As the algorithms and keys are already negotiated we only need to provide a suitable index to the correct security association. The negotiated security association is indicated with a security parameter index (SPI) value. The signature could of course host a public key based signature, but we take advantage of our key exchange procedure so that we are able to use the negotiated keying material to calculate the message authentication code.

We also want to provide a unique identifier *jti* to the protected data objects and it could be used as a sequence number if one wants to avoid potential replays of messages. Note that in case of limited devices, having an increasing number might be preferable option because it is not feasible to keep track of all the received id numbers.

Another use case for the protected data objects is the possibility to encrypt the message. Such example is given in Listing 2.2. The header format follows the same principles as the earlier signed version. The initialization vector and the encrypted payload are given separately, as per JWE [12].

Listing 2.2. Encrypted message header

```
{
  "typ": "JWE",
  "alg": "dir"
  "enc": "A128GCM"
  // index the security association
  "kid": 1201000001,
  "jti": "121213"
}
```

Whenever data objects are protected with symmetric keys they hold value only to the parties who have the corresponding keys. Therefore, a gateway should serve data protected with different keys as separate resources. key wrapping and content encryption keys are used [12].

4 Applicability of Existing Key Exchange Protocols

Rather than completely devising a new key exchange mechanism, we have decided to investigate the feasibility of applying existing protocols.

4.1 DTLS Delayed Key Exchange and Its Security Analysis

This section investigates the applicability of Datagram Transport Layer Security (DTLS) [8] as the base protocol for the delayed key exchange.

DTLS adds a DoS protection mechanism to regular TLS, which makes sure that the connecting addresses are return routable. This exchange is optional, and for sleepy sensors it can be expected that the gateway does this stateless cookie exchange on behalf of the sensor. Because ServerHello message in DTLS contains the server nonce, the gateway cannot proceed in the handshake any further.

Yet, if the DTLS implementation of the sensor is modified slightly, a supporting gateway is able to reduce the key exchange messages to the sensor to two flights. Hartke and Bergmann [9] suggest that retransmitting ClientHello with a statelessly verifiable ServerHello to the responder enables it to delay establishing state until it receives the ClientKeyExchange message. We assume that the gateway is able to respond with a valid ServerHello, and so it can transmit all these messages to the sensor at once. This is shown in Fig. 2.

The user and the gateway complete the first handshake flights. After waking up and sending a fetch message (Fig. 2), the sensor receives the pending handshake messages. After it extracts the premastersecret from the ClientKeyExchange message, it sends ChangeCipherSpec message as well as a Finished message, which uses the derived mastersecret and authenticates the handshake. It should be noted that the messages may need to arrive or be processed in the right order, as implementations, for example Californium [24], expect to be able to derive the master secret when processing the ClientKeyExchange message.

The returning ServerHello is marked as optional in Fig. 2 as the sensor is able to construct it. Indeed, the sensor should have the previously chosen nonce stored,

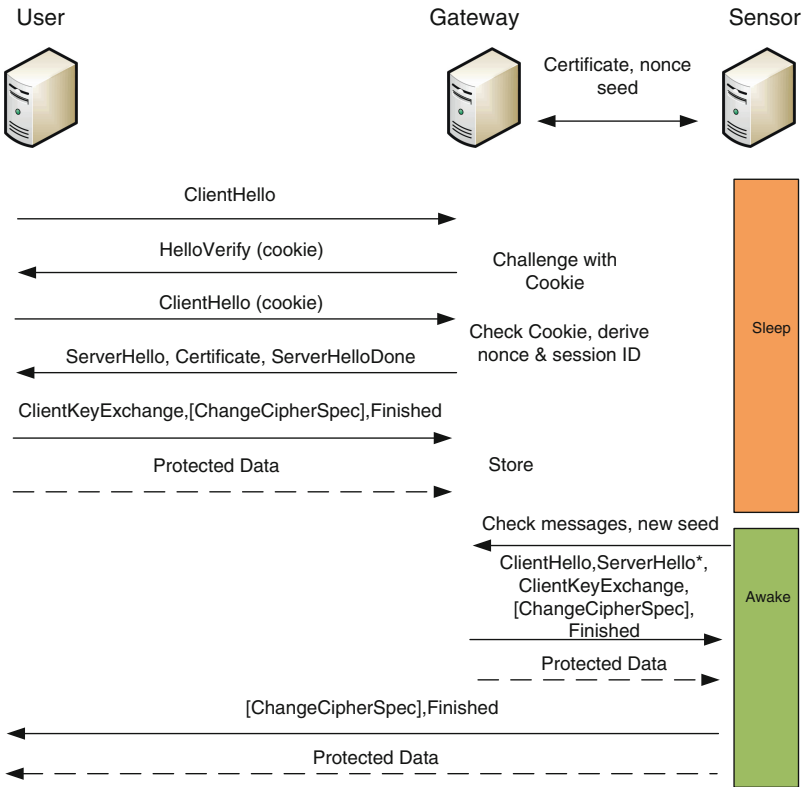


Fig. 2. Message exchange with delayed DTLS

and the Cipher Suite selection is deterministic. However, this restricts the amount of delayed handshakes per sleep cycle to one. As the gateway is not expected to perform DoS, the ClientHello message with a correct cookie is sufficient proof of the return routability of the initiator address. It should be noted that the DTLS return routability cookie does not protect against local DoS, and the gateway or any other node in the wireless network is able to send spoofed initiator packets with correct cookies.

From a security standpoint, the only change to the basic TLS handshake is in the responder's timing, as the ServerHello is transmitted before the ClientHello. This timing change cannot break the security of the key exchange protocol under the indistinguishability assumption [25]. It also does not weaken the sensor's contribution to the resulting symmetric key, if a stored ServerHello is used, or when the used pseudo random function is secure if the gateway derives the nonce.

Indeed, even though the gateway cannot be trusted to choose the TLS server nonce, it could reasonably use a pseudo random function to derive it from a renewed nonce using the client information as a seed. This is why the message in the Fig. 2 is marked as nonce seed, and not as ServerHello, which is the simplest option. Deriving the ServerHello at the gateway enables multiple handshakes over a single sleep-wake cycle. However, such behavior requires tracking of the used ServerHellos while the nonce is considered to be valid, in order to prevent replay attacks.

As noted, the ServerHello message in Fig. 2 is optional. Therefore, also the session ID has to be sent to the gateway in advance. Another possibility is that the gateway derives it in a deterministic fashion. Indeed, if the sensor does not support re-negotiations, the sensor simply sends the session ID to the gateway before the handshake. Yet, if re-negotiations are required, the situation becomes more complex. For re-negotiations the gateway and the sensor could either use the ClientHello session ID as it is, or apply some converging transformation to it (binary AND, for example).

Because both the sent and received handshake messages are authenticated in (D)TLS [5, p.61], the sensor has to recreate all the messages that the gateway has sent. This verifies that everything went according to its preferences. However, these messages need not be sent. Indeed, as there is a valid message in the receive buffer for the next flight, the sensor may transition directly to the next state. Such implementation would still be compatible with a traditional handshake.

By using TLS next protocol negotiation [26] the user can start using the security association even before the handshake is fully completed (Fig. 2). This minimizes the waiting time for the actual communication on the application layer but requires that the user's DTLS implementation supports this extension. The DTLS implementation needs to also expose the capability for applications.

Indeed, DTLS is expected to work close to application layer. Thus, application layer messages can be wrapped into the protected DTLS records and those records can be transferred end-to-end without having to devise a special format for the individual data objects. However, as mentioned the possible upper layer protocols cannot be translated in between without decrypting the records first,

i.e., one cannot first run data objects on top of HTTP and then expect to switch to CoAP, unless one were to make more exotic protocol stack choice, such as data objects over DTLS over HTTP/CoAP.

Using DTLS derived keys with data objects as described in Sect. 3.4, requires an interface to the negotiated keys. However, there already is work concerning exporting keys from TLS and DTLS [27] to be used in the application layer, for example, DTLS-SRTP [28] uses this mechanism for its key derivation.

4.2 Case IKEv2

In this section we investigate the possibility of employing Internet Key exchange version 2 (IKEv2) [4] to implement the suggested key exchange procedure. While the “full” IKEv2 has been used with IPsec security suite for protecting IP traffic, we also consider the “light” version of IKEv2 [29], which might be more suited for limited environments. The minimal IKEv2 supports only limited amount of exchanges and optional features have been mainly left out [29], even though the draft specification does not consider so much suitable algorithmic options. However, it assumes that the device initiating the communication is a limited one, while our architecture has the opposite use case. One should also note that IKEv2 runs on top of UDP.

The pre-configuration phase expects that the sensor informs the gateway about its public key and Diffie–Hellman (DH) parameters. A very minimal sensor implementation might just have one DH group, but a more complex one could use several ones. The gateway just has to make sure that the sensor knows which one has been used in the messages it stores.

IKEv2 negotiation in its basic form consists of two different exchanges: IKE_SA_INIT and IKE_AUTH. They are shown in Fig. 3 as Ix, Rx pairs. The first actual message contains the typical IKE header with a chosen Security Parameter Index (SPI) and the suggested ciphers, which the user is able to use for the establishment of the IKE security association. The message also contains the Diffie–Hellman parameters and a random nonce. In Fig. 3, we have added the cookie mechanism to mitigate DoS concerns, even though IKE minimal [29] leaves that out. The message is re-sent with the cookie attached, in order to prove the return routability of the initiator’s address.

After the cookie is validated, the gateway sends the chosen cipher, another random nonce, and the Diffie–Hellman parameter of the sensor. The gateway needs to select a suitable SPI_s on behalf of the sensor and use a Nonce_s that the sensor has provided to it. The used values need to be communicated to the sensor later, if they are not fixed. The same security considerations for the nonce are needed as in DTLS. In addition, an optional CERTREQ could be used to inform about the trust anchors the sensor is willing to trust.

The next pair of messages (I2 and R2) can be protected using the negotiated cipher and the keying material generated with the help of the nonces and the common DH secret. The initiator’s message contains its identity, which can be, e.g., IP address, but a more interesting possibility is to base it on a public key. That can be used with AUTH parameter to prove the ownership of the

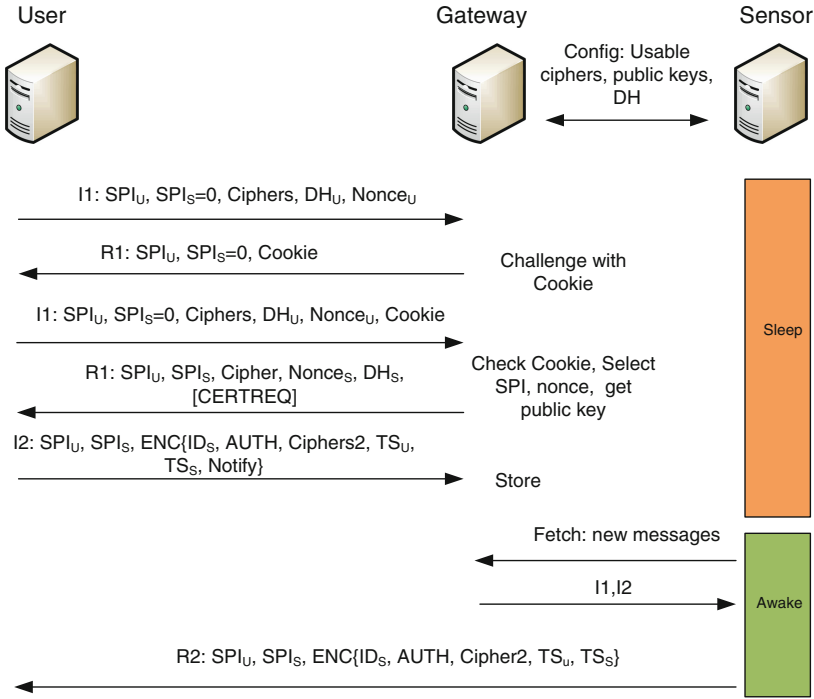


Fig. 3. Message exchange with IKE

key by signing data (basically the received message with some additional data). Alternatively, one could authenticate based on a shared secret.

The message also contains proposal for algorithms, which are to be used for the child security association, i.e., the one that is going to protect data payloads. The Notify parameter in Fig. 3 is used as in [29] to indicate that this is the first association to be created, thus effectively deleting any previous ones between these participants.

Next the I2 message is stored by the gateway to wait for retrieval. When the sensor wakes up and sends a fetch message, the gateway sends both I1 and I2 messages to it. Even though, the sensor might not worry about downgrade attacks, for example, because of limited crypto options, the I1 message is still necessary in IKE, as it is needed for the symmetric secret used in the IKE_AUTH message.

As stated before, the sensor has to support using some values that were used in the gateway's IKE_SA_INIT-reply (message R1), like the aforementioned SPI_S and $Nonce_S$. The sensor will then send the IKE_AUTH message (R2 in Fig. 3) using the values that the gateway has used previously. This message states the selected cipher and the traffic selectors. After that, protected application level messages can be sent.

5 Discussion

It should be noted that while key exchanges are done in the traditional Internet to protect communication flows, it may not always be preferable for constrained nodes. This is true especially for message authentication done by a constrained device. For instance, storing multiple symmetric keys and maintaining sessions can be a burden for constrained devices if the associations are many. Moreover, public key cryptography has other advantages for message authentication. For example, non-repudiation is present when public key based digital signatures are used. Additionally, the signatures can be verified by anyone who has the public key, including the gateway in the presented scenario. This is in contrast to symmetric secrets where, in order to authenticate the same payload for multiple receivers the sensor may have to calculate a separate message authentication code for every recipient.

However, exchanged keys are a natural choice for confidentiality protection. For instance, the amount of stored key material is less for the symmetric keys, as symmetric keys are smaller for the same security level. While public keys could perhaps be stored somewhere else in an authenticated directory and the symmetric keys could be wrapped, some form of key exchange is needed in cases where the recipient lacks an authenticated public key or the liveness of the other party is a concern.

The biggest benefits from the delayed key exchange materialize when the key exchange can be made asynchronous, and amount of transmitted data in the sensor's access network can be minimized. Certificates and certificate chains are lengthy messages, so schemes using certificates for authentication, such as DTLS, are more efficient if the gateway or the infrastructure is able to transmit such messages.

It is also important to minimize the sensor's waiting time for external responses, as denial of sleep can be a concern. Because a normal handshake lasts multiple flights, a sleepy sensor might need multiple sleep cycles to complete it. Moreover, direct key exchange might require the sensor to be awake at the same time the user is accessing it. For the first message this requires either synchronisation and knowledge about the sensor's wake up times, or constant polling. If the gateway would store only the first message and pass through the others, it would have to be aware of the wake up times or the sensor would have to poll the subsequent messages.

The presented key exchanges would both work in the delayed key exchange scenario. These protocols have traditionally been used for different purposes and direct comparison may not make that much sense. However, from the further development point of view, one can consider which is the more promising target. Overall, if we consider the current adoption, and integrating with existing clients, the best choice might be DTLS. For example, IETF CoRE Working Group mainly focuses on DTLS [6] for CoAP. Moreover, DTLS already has some support for using negotiated keys in applications. In addition, the TLS next protocol negotiation enables the client to send its first payload even before the sensor has woken up.

CoAP specification defines an application profile for DTLS, which includes two ciphersuites: PSK based ciphersuite¹, and a ciphersuite based on Ephemeral Elliptic Curve Diffie–Hellman key agreement² [6]. The ephemeral public key ciphersuite signs the initiator’s nonce in an early phase of the handshake, and is not as such compatible with the presented procedure. However, the same primitives can be combined to produce a non-ephemeral version of the ciphersuite, which is compatible with our proposal. Also, the suggested PSK based ciphersuite should work in the presented scenario, particularly if the gateway is able to give identity hints, if necessary.

The security assumptions in the presented key exchange protocols are not broken by the suggested modifications as these protocols are not sensitive to timing changes. There could be benefit in offloading more functions to the gateway, for instance, the integrity calculation in DTLS is heavy. This would avoid the need for the sensor to receive and store all the handshake messages. However, an adversary would then be able to, for example, change the ServerHello message, and control the chosen ciphersuites. Also, the aforementioned replay attacks would be possible if the gateway would be allowed to choose the nonce, for example.

6 Conclusions

In this paper we have proposed a delayed key exchange method, which would take into account the sleeping nature of sensors and use an intermediary gateway to cache key exchange messages for later retrieval. Even though the gateway is used, the suggested method would still be able to provide end-to-end security. In our case, the keys are intended to be used to secure application layer messages, that is, data objects carried over HTTP and CoAP.

We have presented how the suggested procedure would work with two different existing key exchange protocols. Both investigated protocols can be used to do a key exchange in a delayed fashion. Moreover, the procedure is opaque for a client initiating the key exchange. However, some configuration and modification can be needed in the constrained node and the gateway element.

Part of our proposal is the data object security, for which we suggest using JSON based security tokens. Those tokens get their security material from the key exchange, which has been run prior to actual data exchange. JSON provides suitably light weight option and enjoys already widespread adoption in the web world, thus facilitating the integration of the Internet and Internet of Things.

Acknowledgements. The research was conducted in the Internet of Things program of DIGILE (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT), funded by Tekes.

¹ TLS_PSK_WITH_AES_128_CCM_8.

² TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8.

References

1. Giusto, D., Lera, A., Morabito, G., Atzori, L.: *The Internet of Things*. Springer, New York (2010)
2. Garcia-Morchon, O., Keoh, S., Kumar, S., Hummen, R., Struik, R.: Security Considerations in the IP-based Internet of Things. Internet-Draft draft-garcia-core-security-04, Internet Engineering Task Force, March 2012, Work in progress
3. Castellani, A., Loreto, S., Rahman, A., Fossati, T., Dijk, E.: Best Practices for HTTP-CoAP Mapping Implementation. Internet-Draft draft-castellani-core-http-mapping-05, Internet Engineering Task Force, July 2012, Work in progress
4. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P.: Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), September 2010, Updated by RFC 5998
5. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008, Updated by RFCs 5746, 5878, 6176
6. Shelby, Z., Hartke, K., Bormann, C., Frank, B.: Constrained Application Protocol (CoAP). Internet-Draft draft-ietf-core-coap-11, Internet Engineering Task Force, July 2012, Work in progress
7. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security. RFC 4347 (Proposed Standard), April 2006, Obsoleted by RFC 6347, updated by RFC 5746
8. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012
9. Hartke, K., Bergmann, O.: Datagram Transport Layer Security in Constrained Environments. Internet-Draft draft-hartke-core-codtls-02, Internet Engineering Task Force, July 2012, Work in progress
10. Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., Granqvist, H.: Oasis ws-trust 1.4. Specification Version 1 (2008)
11. Shelby, Z.: Embedded web services. *IEEE Wirel. Commun.* **17**(6), 52–57 (2010)
12. Jones, M., Rescorla, E., Hildebrand, J.: JSON Web Encryption (JWE). Internet-Draft draft-ietf-jose-json-web-encryption-05, Internet Engineering Task Force, July 2012, Work in progress
13. Sethi, M., Arkko, J., Keranen, A.: End-to-end security for sleepy smart object networks. In: 2012 IEEE 37th Conference on Local Computer Networks Workshops (LCN Workshops), pp. 964–972. IEEE (2012)
14. Bianchi, G., Caposelle, A.T., Mei, A., Petrioli, C.: Flexible key exchange negotiation for wireless sensor networks. In: Proceedings of the Fifth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, WiNTECH '10, pp. 55–62. ACM, New York (2010)
15. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. *Commun. ACM* **21**(12), 993–999 (1978)
16. Vial, M.: CoRE Mirror Server. Internet-Draft draft-vial-core-mirror-proxy-01, Internet Engineering Task Force, July 2012, Work in progress
17. Kadyk, D., Fishman, N., Seinfeld, M., Kramer, M.: Negotiating secure connections through a proxy server, 7 February 2006, US Patent 6,996,841
18. Ylitalo, J., Melén, J., Nikander, P., Torvinen, V.: Re-thinking security in IP based micro-mobility. In: Zhang, K., Zheng, Y. (eds.) *ISC 2004*. LNCS, vol. 3225, pp. 318–329. Springer, Heidelberg (2004)
19. Dolev, D., Yao, A.: On the security of public key protocols. *IEEE Trans. Inf. Theor.* **29**(2), 198–208 (1983)

20. Nikander, P., Arkko, J., Ohlman, B.: Host identity indirection infrastructure (hi3). In: Proceedings of the 2nd Swedish National Computer Networking Workshop SNCNW 04, 1–4 (2004)
21. Arkko, J., Kernen, A.: CoAP Security Architecture. Internet-Draft draft-arkko-core-security, Internet Engineering Task Force, July 2011, Expired
22. Ylitalo, J., Salmela, P., Tschofenig, H.: Spinat: Integrating ipsec into overlay routing. In: First International Conference on Security and Privacy for Emerging Areas in Communications Networks, SecureComm 2005, pp. 315–326. IEEE (2005)
23. Jones, M., Bradley, J., Sakimura, N.: JSON Web Signature (JWS). Internet-Draft draft-ietf-jose-json-web-signature-05, Internet Engineering Task Force, July 2012, Work in progress
24. Jucker, S.: Securing the constrained application protocol (2012)
25. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
26. Langley, A.: Transport Layer Security (TLS) Next Protocol Negotiation Extension. Internet-Draft draft-agl-tls-nextprotoneg-04, Internet Engineering Task Force, May 2012, Work in progress
27. Rescorla, E.: Keying Material Exporters for Transport Layer Security (TLS). RFC 5705 (Proposed Standard), March 2010
28. McGrew, D., Rescorla, E.: Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP). RFC 5764 (Proposed Standard), May 2010
29. Kivinen, T.: Minimal IKEv2. Internet-Draft draft-kivinen-ipsecme-ikev2-minimal-00, Internet Engineering Task Force, February 2011, Expired



<http://www.springer.com/978-3-662-46337-6>

Ad-hoc Networks and Wireless

ADHOC-NOW 2014 International Workshops, ETSD,
MARSS, MWaoN, SecAN, SSPA, and WiSARN, Benidorm,
Spain, June 22--27, 2014, Revised Selected Papers
Garcia Pineda, M.; Lloret, J.; Papavassiliou, S.; Ruehrup,
S.; Westphall, C.B. (Eds.)
2015, XV, 318 p. 145 illus., Softcover
ISBN: 978-3-662-46337-6