Secure Data Exchange for Computationally Constrained Devices

Vassilis Prevelakis[1], Mohammad Hamad[1], Jihane Najar[1], and Ilias Spais[2]

[1] Institute of Computer and Network Engineering, Technical University of Braunschweig, Germany
{prevelakis, hamad, jihanen}@ida.ing.tu-bs.de
[2] AEGIS IT RESEARCH UG, UK
hspais@aegisresearch.eu

Abstract. The need for secure communications between IoT devices is paramount. However, due to computational constraints and the need for lightweight publish/subscribe type of communications, traditional mechanisms for secure communications such as TLS and IPsec cannot be used directly. In this paper we present a new model for secure exchange of information that is based on off-line symmetric key encryption of the data and the use of policy-based credentials for the release of the encryption keys. Using the mechanism presented in this paper, the sender does not need to establish a direct network connection with the receiver, but can employ a store-and-forward transmission method. End-to-end data security is achieved by encrypting the data before the transfer, and sending the encryption key via an Aegis Secure Key Repository (ASKR) server. By encrypting the data off-line, the transmission can be carried out as fast as possible.

Keywords: Secure Communications · Trust Management · Policy-based Credentials · IoT Security · Aegis Secure Key Repository.

1 Introduction

In 2007 the British tax service lost 2 CDs containing 25 million tax records [1]. More recently, a laptop stolen from NASA contained command codes used to control the International Space Station [10]. If the data on the CDs or laptop had been encrypted, the whole incident would have been a non-issue. When dealing with confidential data (e.g. biometrics) exchanged by IoT devices the situation becomes more serious, as these exchanges are often over insecure over-the-air channels and the devices involved are in most cases resource constrained and hence are not in a position to employ traditional security protocols such as TLS and IPsec.

IoT infrastructures, because of their heterogeneity and the different accelerators used, are susceptible to various threats, each targeting a different component. Roman et al. present an extensive analysis of the security threats and

challenges in an edge computing architecture [19], explaining that security issues might arise at different levels: Network infrastructure (Denial of Service, Man in the Middle), Edge (Physical damage, Privacy leakage, Privilege escalation, Service manipulation, Rogue data centre), Virtualization Infrastructure (Misuse of resources, Privacy leakage, Privilege escalation, VM manipulation), and User Devices (Injection of information, Service manipulation).

Common security properties include confidentiality, authentication, integrity, authentication (source or mutual), and availability. Unfortunately, the IoT domain places additional constraints, such as low power [16], making it very difficult to ensure security [2]. In 2014, the Defense Advanced Research Projects Agency (DARPA), stated that achieving IoT security will have a potential impact broader than the Internet itself [21]. Examples of security failures in IoT systems abound [16], [15] and have been shown to have adverse impact on both user confidence as well as tangible financial loss, even potentially life threatening [11].

As mentioned above the reduced capabilities of IoT devices both in the power consumption front as well as the overall availability of processing power necessitates the use of light-weight security mechanisms. To make matters worse, despite the limited power available to each IoT device, ganging them together in a coordinated attack can have devastating consequences against the victim. The IoT devices themselves may become victims of DOS attacks, preventing them from carrying out their tasks. Seepers et al. [20] introduced a novel technique designed at implantable medical devices (IMDs), whereby the initiator of the communication provides the power to perform a first level of authentication, so as to prevent an attacker from running down the battery of the IMD through bogus connection attempts. Another technique used in the HIP protocol [14] involves the solution of a puzzle before any processing is allowed to happen in the IoT device, thus forcing the attacker to expend resources in mounting an attack.

Communication is a common function of IoT devices and needs to be protected. In order to achieve this, we need to consider existing Internet security protocols. IKEv2/IPsec [9], TLS [17], DTLS [18], HIP ([13], PANA [5], and EAP [23], as well as the Constrained Application Protocol (CoAP) [22], which is tailored to IoT devices.

IKEv2/IPsec was supposed to be the standard security protocol and would be an integral part of IPv6, but lack of an effective API for establishing connections as well as incompatible implementations restricted its use to VPNs. A key complaint against IPsec is that an application cannot be sure whether its connection with a remote peer is protected or not. The less used Host Identity Protocol (HIP), like IKEv2, performs an authenticated key exchange in order to set up the IPsec security associations. TLS proved a more convenient security protocol as it is application driven. TLS requires TCP and hence is bound to a connection-based reliable protocol. On the other hand DTLS uses datagrams and can run over UDP. Both TLS and DTLS are kept similar by design and share the same crypto algorithms. CoAP is designed to work with DTLS. It

is a RESTful protocol, which, while closely resembling HTTP, runs over UDP for added efficiency. It provides three modes: *PreSharedKey* which assumes that both ends have their symmetric key preinstalled, *RawPublicKey* which uses pre-installed asymmetric keys, and, finally, Certificate mode where the public keys are certified by a CA which issues the appropriate certificates. RFC-7925 [6] defines a special IoT implementation profile for both TLS (V1.2) and DTLS (V1.2), that is designed for devices with reduced capabilities.

Some protocols such as IPsec that were designed and implemented decades ago are quite suitable to resource constrained devices [7] since the technological advances have put workstation-class computing power from the late 1990s into modern microcontrollers. At the same time, crypto such as Elliptic Curve Cryptography [12] and stream ciphers such as ChaCha [3] go a long way towards making security protocols more resource efficient.

Thus, it is clear that even devices with severe processing constraints can still carry out sophisticated security functions even though they may not be able to perform them at the same speed as their more capable peers. In addition, power constraints may not allow direct on-line connection between data producer and consumer. This has led to the introduction of publish/subscribe protocols such as MQTT [8]. This model allows the device to contact a nearby server (called the message broker) to deposit the information it wishes to send. Any number of consumers of this data may then download it from the broker.

In particular, our example application consists of a large number of IoT imaging sensors that need to make decisions that depend on the analysis of the captured images. However their limited processing capabilities prohibit local processing of the images. Instead each device follows a preset cycle whereby it wakes up takes a sample, sends the sample for processing and goes back to sleep. When it wakes up again it collects the processed data, makes its decision, takes another sample, sends it for processing and goes back to sleep. This cycle allows the device to power up its wireless transmission hardware only when it has something to send. As such it is a push-only device and cannot accept incoming connections. If the architecture relied on a small number of fixed image processing servers, then each IoT device would have the necessary credentials pre-stored and open a secure connection with one of these servers. Simple load-balancing schemes, such as DNS-based round robin, would spread the load among the servers. However, our architecture depends on the ability to use any server from a dynamic pool requiring a more flexible solution.

Moreover, we need to ensure the integrity and confidentiality of the data while in transit. Point-to-point schemes such as those described above do not provide end-to-end security, so we need to encrypt the data before it leaves the sender. Any bulk encryption algorithm can be used but we still need to address the problem of sending the encryption key to the consumer, especially if they are more than one, and their identity is not known a priory. Hence we arrive at the basic requirements for a secure data exchange protocol for computationally constrained devices:

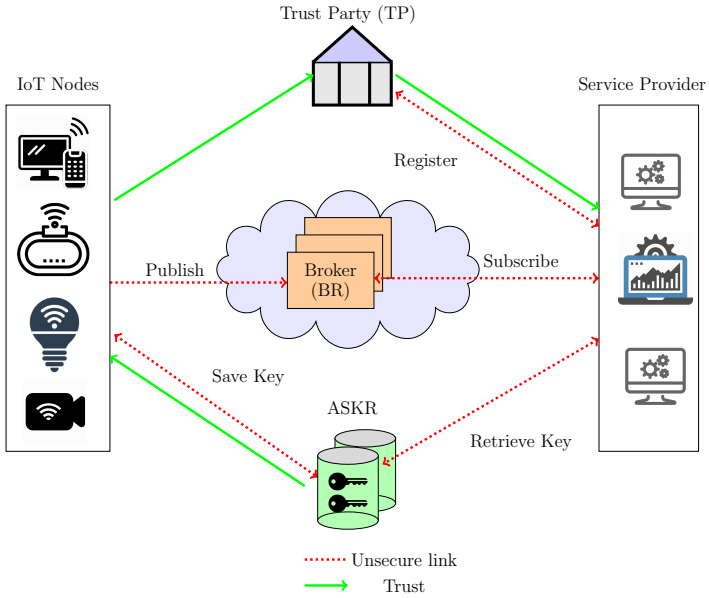

Fig. 1. ASKR System Architecture.

1. Must provide end-to-end data integrity and confidentiality, even if the recipient or recipients are not known in advance.
2. Must support publish-subscribe data exchange mechanisms.
3. Must be compatible with low power devices.
4. Must ensure high availability.
5. Optionally it must support non-repudiation.

In the next section we present the Aegis Secure Key Repository (ASKR) and show how it was adapted to accommodate our data exchange protocol. We then discuss the security implications of the use of our protocol, and present a performance analysis of the system. Finally, we present our closing comments and future plans.

2 System Architecture

Figure 1 presents the proposed system architecture which aim to fulfill the security and functional requirement mentioned in the previous section.

2.1 System components

The components of our system are:

– **IoT Node ($\mathbf{N}_i$)**: which represents the IoT devices which collects the data.

- **Service Provider (SP)** which represents the server who can process and analysis the data of the Iot Nodes. Each one of these servers has certain capabilities make it suitable to process certain requests.
- **The Aegis Secure Key Repository (ASKR)**: ASKR is a key repository that enforces controlled access to data storage keys. Each time one Iot node needs to transmit the data securely, it saves the secret key which is used to encrypt this data within the ASKR. Later, ASKR revives a request from one SP to retrieve that key and use it to decrypt the data before processing it.
- **Trust Party (TP)** which represents the single trust part of the system for the IoT nodes. IoT node trusts TP that it will introduce trustworthy service providers.
- **Broker (BR)** represents the remote server which is used to receive the data from the IoT nodes and store it until one SP proposes to process this data.

It is important to note that all communications among the different components are exchanged over unsecured links. The Detailed information about the exchanged messages are described in the next chapter.

2.2 Key Access Credential (KAC)

A vital component of the system is the access control enforcement mechanism that is embedded within the access credentials to ensure that access control policies (both user-defined and system-wide) are enforced. Each issued credentials is a translation of the trust relationship between the different components by linking their public keys through a trust relation under certain conditions.

Figure 2 shows one example of such credential where N_1 issues this credential to state its trust to the TP. As we explained before TP is used to authenticate on the current service providers which exist in the system. Thus, this credential Implicitly trusts any SP which is trusted by TP and have specific capabilities which included in the credential. To ensure the integrity of this information, The issuer (it is called Authorizer) signs it using its private key.

2.3 Policy Description Language

The policy description language used to develop the credentials enables the specification of fine-grained access control policy for each key. For example, a credential may contain authorization information for multiple users allowing group access control. Also the policy may limit access to the key outside office hours or during holidays. Another benefit is that it can require the consent of multiple users for the release of a key. Credentials may also be used to force re-keying of the storage key. In this section we describe the use of the KeyNote [4] policy definition language within our system, providing examples of a number of scenarios involving delegation, group access, RBAC, etc.

```
Keynote-Version:2
Local-Constants:
        N1 = "rsa-hex:308189028181009be7f55296c ... "
        TP = "rsa-hex:30818902818100c497d6da ... "
        DEV_GRP_KEY = "rsa-hex:30818902818100cafd3b15ae93b4 ..."
Authorizer: N1
Licensees: TP
        Conditions: app.domain == "ASKR"
        && key-oid == 0x931d2c6792df
        && ( SP_capability == cp1) - > "true";
Signature: "sig-rsa-sha1-hex: 526ded0b90b91f2898778de85e4b7 ... "
```

Fig. 2. Sample Key Access Credential.

Delegation On of the main advantage of KeyNote policy definition language is trust delegation. ASKR requires that each request is signed by the requester and that the credentials supplied by the requester describe a path from the requester's key to a key that ASKR trusts. Therefore, we can use additional credentials to extend the number of servers that can request a storage key. Consider the case where one service provider (SP_1) wishes to allow another one (SP_2) to check out a storage key of one of file of a certain IoT node. She issues a credential that delegates access to this key to new server key. This will allow SP_2 to access the file independently of SP_1 (See Figure 3).

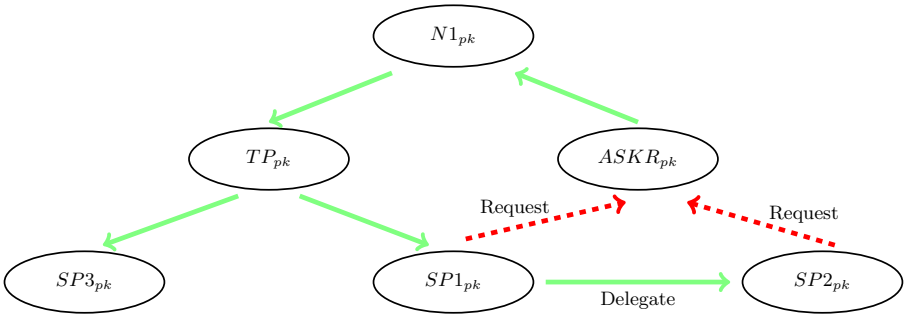

Fig. 3. Access Credential Delegation Chain.

3 Proposed Protocol

In this section, we will explain the different phases of our proposed protocol between the component of the system. Within this section, we represent the Encryption operation of message M using a key K as $ENC_K(M)$, we also represent the signing operation of message M using a key K as $SIG_K(M)$. Also, we refer to the public key of Component C as C_{pk} and to its private key as C_{pr}. Finally, we refer to the KAC from Component $C1$ to Component $C2$ as $(Cr_{C1_{pk}2C2_{pk}})$

3.1 Phase 0: Service Provider Registration

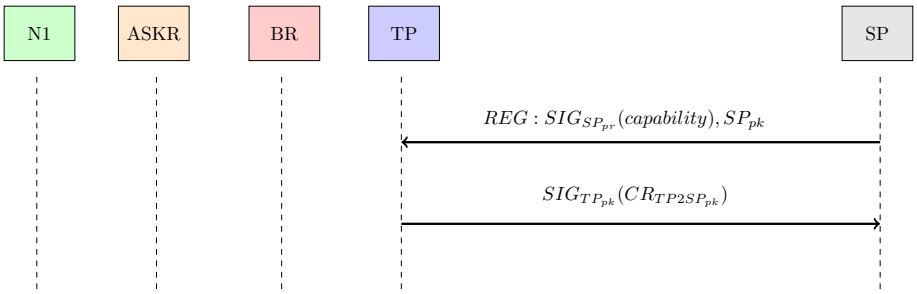

Fig. 4. Service Provider Registration

This phase occurs only whenever a new service provider attends the systems. Each new SP needs to register its self with the Trusted Party by following the next steps (see Figure 4):

- At first, the SP Ask the TP to register (authenticate) herself by sending its capabilities signed by its private key (SP_{pr}). Besides, she sends its public key (SP_{pk}) to be used by the TP during the validation of the signed request.
- After validating the signature, TP creates a KAC ($Cr_{TP_{pk}2SP_{pk}}$) which authorize the requested SP to process data published by IoT nodes which trust TP.

Someone could say that any SP can pretend to have faked capabilities. To prevent that, each SP needs to provide a signed credential(s) from a common trust authority between the TP and the SP (cross-domain trust relationship). This credential should include information which supports its pretension. However, to make the system simple here, we assume that each SP is honest during the registration process.

3.2 Phase 1: Saving the Key

This phase represents the actual first step in the protocol. It happens whenever an IoT node (let say N1) wants to publish data. N1 needs to follow the next steps (see Figure 5):

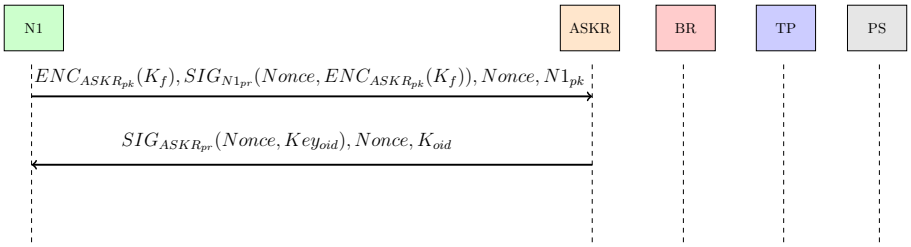

Fig. 5. Saving the Key

- N1 create the key (K_f) which will be used as a symmetric key to encrypt and decrypt the data.
- N1 uses the ASKR public key ($ASKR_{pk}$) to encrypt the key before sending it to ASKR prevent anyone from extracting it from the message.
- Then, N1 signs the encrypted key within its private key ($N1_{pr}$) and send it along with its public key ($N1_{pk}$). Note that, Nonce is used to prevent a replay attack.

Note that there may be many ASKR servers, so N1 needs only to chose the first available one to save the key there. We Assume that the ASKR will stay available for the next period, which is required for one SP to contact it and get the key. Now, Whenever the message arrives at the ASKR, it exercises the next steps:

- The ASKR uses the $N1_{pk}$ to authenticate N1 and to verify the integrity of the transmitted message.
- Then, it uses its private key ($ASKR_{pr}$) to extract the key (K_f) by decrypting the message.
- ASKR creates an entry includes the $N1_{pk}$ and (K_f) as well as an identifier value for this entry called Key_{oid} (it could be the hash value of both keys) and save it within its data store. This identifier will be used later by to reach to this entry.
- ASKR signs the Key_{oid} along with a nonce and sends them to the N1.

It is important to note that, instead of sending Key_{oid} to N1, ASKR can create a credential ($Cr_{ASKR_{pk}2N1_{pk}}$) which authorizes N1 and, implicitly, any component trusted by N1 to get the key. However, saving the $N1_pk$ eliminates the need for that.

3.3 Phase 2: Publishing the Data

In this phase, the N1 become ready to transfer the data to the broker. Figure 6 shows the required steps :

- N1 uses the generated key (K_f) to encrypt the data locally.

– Besides, N1 creates KAC ($Cr_{N1_{pk}2TP_{PK}}$) to authorize any SP trusted by TP with specific capabilities to get the data and encrypted it. This KAC is shown in Figure 2. The Key_{oid} is included in this KAC as well as information about where an SP can find the key (e.g., IP of ASKR).
– Then, N1 combines the KAC with the encrypted data and send them to the BR.

Note that, certain file systems (NTFS, Solaris, etc.) allow us to associate additional information with each file so we can store the KAC along with the transmitted file (data) in a transparent manner. We do not need to encrypt the KAC since it does not include secret information. Also, since it is already signed, we can guarantee that no one will be able to change any part of it.

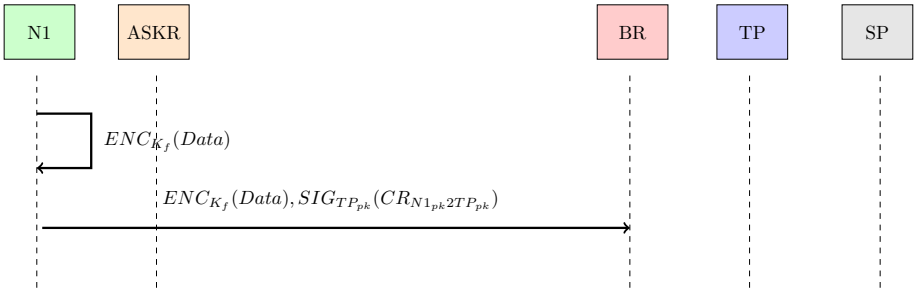

Fig. 6. Publishing the Data

3.4 Phase 3: Subscribing the Data

This phase, which is shown in Figure 7, represents the most simple part of the protocol.

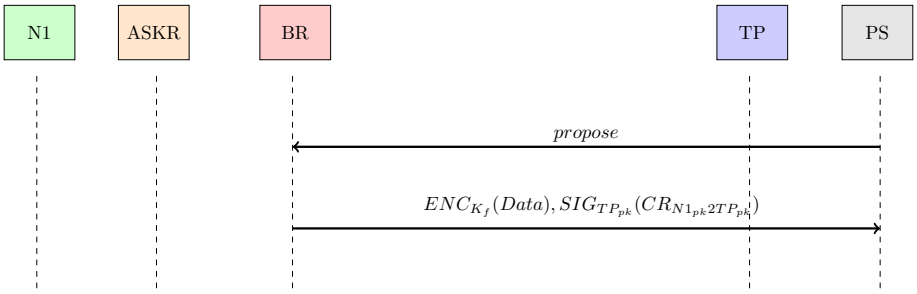

Fig. 7. Subscribing the Data

- Whenever the data is saved in BR, an SP will propose to process it by sending a request tp BR.
- BR Send the encrypted data as well as the received KAC to the SP.

We assume that BR will give the data to any SP who requests and not depend on only the first one request it however, DoS attack need to be considered and prevented. SP who gets the file without having the required capabilities and the valid KACs will not benefit from the data since it is encrypted and it is impossible for her to get the key from ASKR.

3.5 Phase 4: Retrieving the Key and Decrypting the Data

This is the last phase of the protocol. During this phase, the next steps take place (see Figure 8):

- SP extracts the Key_{oid} and IP of ASKR from the KAC ($Cr_{N1_{pk}2TP_{pk}}$) which was received along with the encrypted file from BR.
- SP signs Key_{oid} with a nonce using its private key (SP_{pr}) and sends them to the ASKR along with the both KACs that she has (i.e., $Cr_{N1_{pk}2TP_{pk}}$ and $Cr_{TP_{pk}2SP_{pk}}$) and its public key (SP_{pk})
- The ASKR uses SP_{pk} to verify the request. in case it is verified, ASKR uses the received Key_{oid} to determine the entry which includes the required key (i.e., K_f) as well as the node key which created the key (N1_{pk}) which represents the root of trust.
- ASKR checks If SP has the right to receive K_f by looking at the KACs that she provided and trying to find a path of trust linking the SP to the N1.
- And if so, ASKR sends the key to SP after encrypting it using the SP public key SP_{pk}
- SP receives the message, decrypts it using its private key SP_{pr} and extracts the key K_f which will be used to decrypt tha file.

4 Discussion

4.1 Security Considerations

By choosing to store keys in the ASKR rather than data items themselves, we made the ASKR a very lightweight server, compatible with the rest of the IoT environment. The keys and associated unique identifiers are fixed size and hence easy to store in a conventional DBMS. As we mentioned above, the ASKR only stores the keys, not the KACs since these are supplied by the requester. Moreover, since ASKR is geared towards security it makes sense to keep the system as small as possible reducing the attack surface and the amount of code to be audited. Adding file access primitives would increase complexity and code size. Depending on the security-reliability considerations, redundant ASKR servers may be used. Moreover the encrypted data may be sent to multiple servers to ensure the resilience of the entire system to DOS attacks or natural disasters.

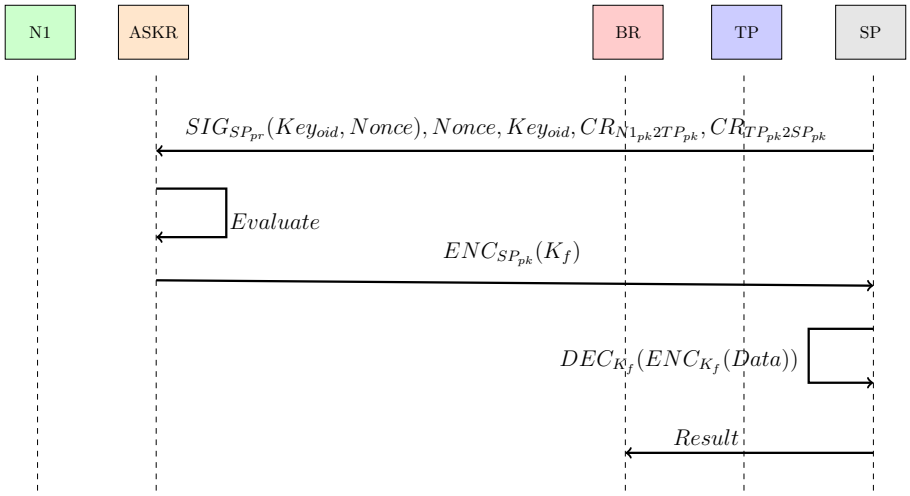

Fig. 8. Retrieving the Key and Decrypting the Data

If ASKR is compromised, the intruder will only have access to the keys, not to the contents of the files. The keys, or key access credentials do not contain any information that may allow the file itself or its contents to be identified. Thus discovering one or more keys is of no use to potential attackers, as they will have to gain access to the encrypted files as well. It is like finding a key on the street; it is of no use unless we know which lock it unlocks. In any case we can split each key and save the parts in multiple ASKR servers using techniques such as key escrow or key shares. In this case multiple ASKR servers will need to be compromised for an entire key to be disclosed.

Note that once someone has access to a key, she can keep a copy forever, thus avoiding the scrutiny of ASKR. This is expected and tolerated by the system since access to a key implies access to the file, so even if we decided to institute some mechanism whereby the key became invalid after a while, then the SP would simply keep a copy of the decoded data rather than the key to the encrypted data. Mechanisms to prevent this behavior fall in the area of Digital Rights Management and are outside the scope of this paper.

4.2 Performance Evaluation

For the performance evaluation, we used a setup that contains two Raspberry Pi computers Model B+ V1 running Raspbian Linux to represent the IoT node and ASKR, and one personal computer running Intel Core i7-7820HQ CPU @ 2.90GHz, with 8 GB of memory (RAM), running on Linux (Ubuntu 16.04 LTS) to represent the SP.

The goal of our measurement was to calculate the required time for sending data via an SSL connection between one Raspberry Pi and the computer and compare it with our proposed system. We measure the time for sending and

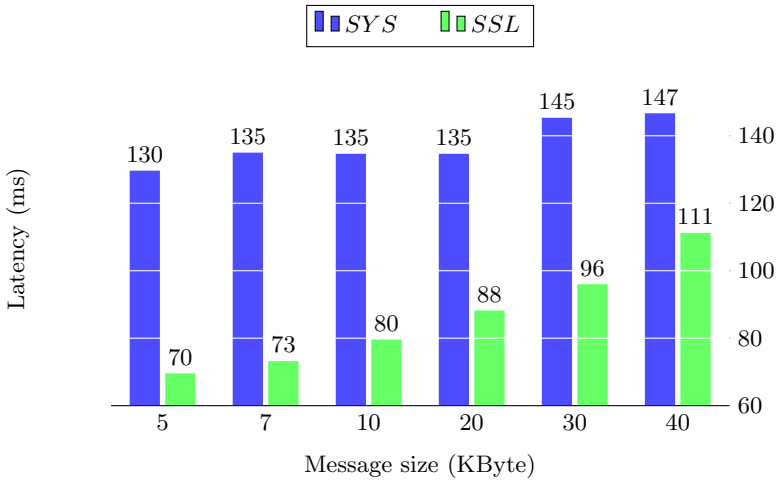

Fig. 9. End-to-end latency between the RPI and Laptop by using SSL and our proposed system.

receiving data from size 5Kbyte to 40Kbyte in both cases. We have repeated the test 3 times for each file size and consider the average of these values.

Figure 9 shows the result of sending data in both cases. As it is clear in the figure, our proposed system is slower than SSL and introduces some overhead.

Figure 10 shows a comparison between the spent time during the encryption and the sending operation for different file sizes on the IoT node. As we can see in the figure,the required time to encrypt the tested files varied from 2 to 6 milliseconds. While it takes 5 to 22 milliseconds to send them. Based on that we can say that by using our proposed system and encrypt the file off-line we can save a good amount of the time compared to the case when we encrypt the data online (i.e., via SSL). Since the use of network represents one of the parts which cause a massive power consumption within the IoT device, we can conclude that by using our system we can save much power by running the network as minimum as required.

4.3 Benefits

Benefits of the use of the ASKR method:

- There is no need for a direct connection between N_i and SP_i. The BR can forward the data to the first available SP without the need to consult with N_i.
- Bulk encryption is done offline. Hence devices without a lot of processing capabilities can do the encryption at a pace that does not strain their energy budget. Also, as there are no timeouts, the process can take as long as necessary.

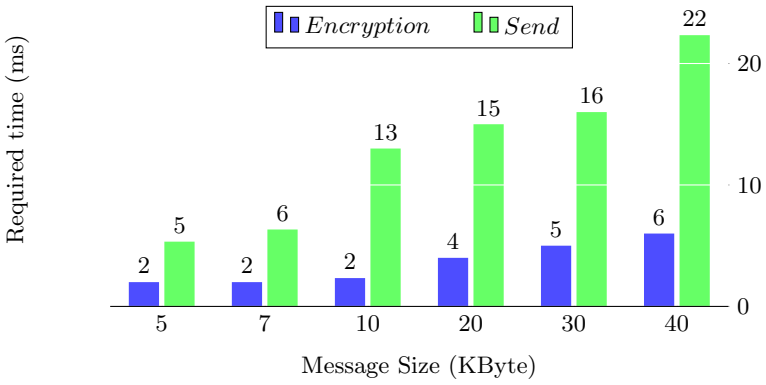

Fig. 10. Required time for executing encryption and sending different files on the IoT node

- Node N_i sends the data to its nearest neighbor, so there is no need to establish an end-to-end connection with a remote server for the data transmission.
- Node N_i can send its key to an ASKR server without need to setup any secure link with it.
- The ASKR server can only maintains a table with (key_{oid}, Ni_{pk},K_f) tuples, so it has no information on what data corresponds to each key in its repository. Once it receives the KAC from SP, it can evaluate the policy and, if its is permissible to release the key, it can locate it from the key_{oid} in the KAC.

Hence the entire process is both low overhead, employs a lot of redundancy, ensures the confidentiality and integrity of the transmitted data, and is compatible with a store and forward transmission mechanism that is suitable for low power devices.

5 Conclusion

In this paper we presented a novel mechanism that supports end-to-end security for a store-and-forward communications regime. We have shown that the security overhead is minimal even on fairly limited devices such as the Raspberry PIs. By using light-weight encryption algorithms and protocols, we created a system that, on one hand, meets the requirements we presented in the beginning of this paper and, on the other hand, is ideally suited to computationally constrained devices commonly encountered in IoT environments. In particular, by removing the need for the on-line secure transmission of the bulk data, we allow devices without direct connections to the Internet to send data to remote servers. Our future plans include the integration of the DTLS protocol in the communication with the ASKR server and the use of a standardized publish/subscribe protocol such as MQTT.

ACKNOWLEDGMENTS

This work is partially supported by the European Commission through the following H2020 projects: THREAT-ARREST under Grant Agreement No. 786890, I-BiDaaS under Grant Agreement No. 780787, CONCORDIA under Grant Agreement No. 830927, C4IIoT under Grant Agreement No. 833828, and SmartShip under Grant Agreement No 823916.

References

1. Risks digest 24.90. http://lists.jammed.com/RISKS/2007/11/0001.html, accessed: 2019-07-08
2. Alaba, F.A., Othman, M., Hashem, I.A.T., Alotaibi, F.: Internet of things security: A survey. Journal of Network and Computer Applications **88**, 10–28 (2017)
3. Bernstein, D.J.: Chacha, a variant of salsa20. In: Workshop Record of SASC. vol. 8, pp. 3–5 (2008)
4. Blaze, M., Keromytis, A.D.: The keynote trust-management system version 2 (1999)
5. Forsberg, D., Patil, B., Yegin, A.E., Ohba, Y., Tschofenig, H.: Protocol for Carrying Authentication for Network Access (PANA). RFC 5191 (May 2008). https://doi.org/10.17487/RFC5191, https://rfc-editor.org/rfc/rfc5191.txt
6. Fossati, T., Tschofenig, H.: Tansport layer security (TLS)/datagram transport layer security (DTLS) profiles for the internet of things. Transport (2016)
7. Hamad, M., Prevelakis, V.: Implementation and performance evaluation of embedded ipsec in microkernel os. In: 2015 World Symposium on Computer Networks and Information Security (WSCNIS). pp. 1–7. IEEE (2015)
8. ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1 (2016)
9. Kaufman, C., Hoffman, P.E., Nir, Y., Eronen, P., Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (Oct 2014). https://doi.org/10.17487/RFC7296, https://rfc-editor.org/rfc/rfc7296.txt
10. Klotz, I.: Laptop with NASA workers' personal data is stolen. https://www.reuters.com/article/us-space-nasa-security/laptop-with-nasa-workers-personal-data-is-stolen-idUSBRE8AE05F20121115 (NOVEMBER 2012), accessed: 2019-07-08
11. Lowry, P.B., Dinev, T., Willison, R.: Why security and privacy research lies at the centre of the information systems (is) artefact: Proposing a bold research agenda. European Journal of Information Systems **26**(6), 546–563 (2017)
12. McGrew, D.A.: Fundamental elliptic curve cryptography algorithms (2011)
13. Moskowitz, R., Heer, T., Jokela, P., Henderson, T.R.: Host Identity Protocol Version 2 (HIPv2). RFC 7401 (Apr 2015). https://doi.org/10.17487/RFC7401, https://rfc-editor.org/rfc/rfc7401.txt
14. Moskowitz, R., Hummen, R.: Hip diet exchange (dex). draft-moskowitz-hip-dex-00 (WiP), IETF (2012)
15. Petersen, H., Baccelli, E., Wählisch, M.: Interoperable services on constrained devices in the internet of things (2014)
16. Porras, J., Pänkäläinen, J., Knutas, A., Khakurel, J.: Security in the internet of things-a systematic mapping study. In: Proceedings of the 51st Hawaii International Conference on System Sciences (2018)

17. Rescorla, E., Dierks, T.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Aug 2008). https://doi.org/10.17487/RFC5246, https://rfc-editor.org/rfc/rfc5246.txt
18. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Jan 2012). https://doi.org/10.17487/RFC6347, https://rfc-editor.org/rfc/rfc6347.txt
19. Roman, R., Lopez, J., Mambo, M.: Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems **78**, 680–698 (2018)
20. Seepers, R.M., Weber, J.H., Erkin, Z., Sourdis, I., Strydis, C.: Secure key-exchange protocol for implants using heartbeats. In: Proceedings of the ACM International Conference on Computing Frontiers. pp. 119–126. ACM (2016)
21. Sfar, A.R., Natalizio, E., Challal, Y., Chtourou, Z.: A roadmap for security challenges in the internet of things. Digital Communications and Networks **4**(2), 118–137 (2018)
22. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252 (Jun 2014). https://doi.org/10.17487/RFC7252, https://rfc-editor.org/rfc/rfc7252.txt
23. Vollbrecht, J., Carlson, J.D., Blunk, L., Ph.D., D.B.D.A., Levkowetz, H.: Extensible Authentication Protocol (EAP). RFC 3748 (Jun 2004). https://doi.org/10.17487/RFC3748, https://rfc-editor.org/rfc/rfc3748.txt