Secure Software Updates for IoT Based on Industry Requirements

Ludwig Seitz[1], Marco Tiloca[2], Martin Gunnarsson[2] and Rikard Höglund[2]
[1]*Combitech AB, Malmö, Sweden*
[2]*Cybersecurity Unit, RISE Research Institutes of Sweden, Sweden*

Keywords: Security, Software Update, Industrial Control Systems, Internet of Things.

Abstract: This paper analyzes the problem and requirements of securely distributing software updates over the Internet, to devices in an Industrial Control System (ICS) and more generally in Internet of Things (IoT) infrastructures controlling a physical system, such as power grids and water supply systems. We present a novel approach that allows to securely distribute software updates of different types, e.g., device firmware and customer applications, and from sources of different type, e.g., device operators, device manufacturers and third-party library providers. Unlike previous works on this topic, our approach keeps the device operator in control of the update process, while ensuring both authenticity and confidentiality of the distributed software updates.

1 INTRODUCTION

Internet of Things can encompass many types of connected devices or *things*. A class of devices especially worthy of study from a security perspective are Industrial Control Systems (ICS). Connected devices that control processes, e.g., in power grids or factories, must have strict security requirements due to the severe potential damage caused by incidents. Providing updates to vulnerable software (SW) is critical for maintaining a system's security. This is exacerbated in ICS, as their components often cannot easily be updated (Davidson et al., 2018).

An issue with these systems is the very long lifetime of devices (tens of years) and the fact that they cannot simply be rebooted, as often required for SW updates. This leads to deployments where devices with known vulnerabilities keep running unpatched, making life easy for attackers. While attacks by Advanced Persistent Threat groups, e.g., the Stuxnet worm or the Black Energy attack on the Ukrainian power grid, have raised awareness in the industry, a solution is not forthcoming, as it would require wide-ranging replacement of legacy devices.

The International Electrotechnical Commission (IEC) released a report on patch management in ICS, under the IEC 62443 series (IEC, 2015). The report focuses on recommendations to establish sound management procedures, for both operators and manufacturers of ICS devices, e.g., requirements on device inventorying, meta-data on the inventory's SW versions and notification of available updates. The report also includes technical recommendations, on assuring that a patch can be matched to the correct devices. To this end, it defines a *Vendor Patch Compatibility (VPC)* file, which uses XML to represent information about the patch and the issuing vendor, enabling operators to determine which devices are suitable to apply this patch to. Cryptographic protection of the VPC file is explicitly placed out of scope.

At the Internet Engineering Task Force (IETF), the SUIT Working Group (IETF, 2022) has worked on solutions for securing SW updates for IoT devices (Moran et al., 2022). Connected ICS devices can be seen as a subgroup of IoT devices, so this work is highly applicable too. SUIT has identified relevant threats to SW updates, and has specified a manifest file format (akin to the IEC VPC file), also containing meta-data of a SW update. While they partly overlap, the SUIT design explicitly considers possible malicious activities, while the approach based on VPC does not.

This paper presents a novel architectural approach for securely distributing SW updates to devices in an ICS over the Internet, based on requirements from an ICS manufacturer. Our approach is partly based on the work of the IETF Working Group SUIT (IETF, 2022) and the recommendations of IEC-TR-62443-2-3 (IEC, 2015). At a high-level, our update distribution process consists of the following phases.

(1) "Update release". A SW update is distributed to the target device domain, possibly via an *Update*

Seitz, L., Tiloca, M., Gunnarsson, M. and Höglund, R.
Secure Software Updates for IoT Based on Industry Requirements.
DOI: 10.5220/0011790100003405
In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP 2023)*, pages 698-705
ISBN: 978-989-758-624-8; ISSN: 2184-4356

Server where it is stored for later retrieval. A domain is a segment of a network controlled by one entity. The update is authenticated, integrity-protected and encrypted.

(2) "Update fetching". An *Update Dispatcher* in the target device domain obtains the SW update, as well as the key material to decrypt and verify the update. This involves per-domain key material made securely accessible to specific Update Dispatchers.

(3) "Update distribution". The Update Dispatcher securely uploads the SW update to the target devices, with the device operator keeping control of the update process. The devices assert the update authenticity by verifying the signature from the update publisher.

To the best of our knowledge, this is the first approach with all the following benefits. (i) Multiple types of SW updates are supported, e.g., custom applications and device firmware, originated by multiple sources of different types, e.g., device operator, device manufacturer and third-party library providers. (ii) The device operator retains control of the update process, while third-party SW providers can still make their critical updates available. (iii) Besides integrity and source authentication of SW updates, also their confidentiality is ensured, preventing reverse engineering and possible exploitation of vulnerabilities.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 defines the requirements. Section 4 presents our update architecture. Section 5 details the the update process. Section 6 provides a security analysis. Finally, in Section 7, we draw our conclusions.

2 RELATED WORK

Secure Software updates for connected devices is a complex topic. Software updates for ICS and IoT have been studied from a multitude of directions. In (Mugarza et al., 2020) the authors have reviewed IEC-TR-62443 and related standards, with regards to software updates of industrial IoT devices (IIoT). The authors note that the stakeholders; asset owners, service providers, and product suppliers have to address software updates jointly.

The authors of (Hernández-Ramos et al., 2020) describe challenges for software update of IoT devices. The authors have focused on SUIT and academic research utilizing blockchain technology. In (El Jaouhari and Bouvet, 2022), the authors discuss technical solutions such as Secure Elements (SE), and Root of Trust (RoT) and the role of such technology in software updates. The authors provide an extensive overview and evaluation of proposed schemes for software update for IoT. In (Catuogno et al., 2017) the authors propose a system for distribution of encrypted software patches that can only be decrypted on systems that fulfils the requirements.

Several surveys detailing specific parts of the software update process have been published. In (Arakadakis et al., 2021), the authors provide a survey of Firmware over The Air Update (FOTA) mechanisms under 1999-2020. They survey technical challenges for a FOTA protocol as well as of commercial and open-source FOTA applications.

In (Petrov, 2018), a patch delivery infrastructure is proposed for supervisory control and data acquisition (SCADA) systems, based on a use-case from a manufacturer of SCADA equipment. The thesis analyzes the requirements of the manufacturer and its customers, showing significant similarities with those from the ICS manufacturer we have been in contact with. The thesis proposes a multi-layered solution for patch delivery, operated by the manufacturer.

In (Asokan et al., 2018), the authors present ASSURED, a secure update mechanism for embedded devices, also applicable to devices in critical infrastructure. ASSURED does not consider 3rd-party libraries that may be deployed on devices, for which an update process must provide support. Instead, the approach we present in this paper does address both those points.

In (Fassino, 2016), the authors present the secure firmware update mechanism deployed by Schneider Electric. It includes signing of the firmware updates and a cloud-based security service to provide updates to customers. The described mechanism does not elaborate on automating the update procedure, while enforcing update policies by the client.

In (Ambrosin et al., 2014), the authors propose a scheme for software update distribution in *cache-enabled* networks. The scheme utilize symmetric encryption of updates to achieve deduplication of stored updates in a cache. The decryption keys are then distributed using attribute based encryption.

3 REQUIREMENTS

This section lists the requirements for secure SW updates, determined based on three sources: a use-case presented to us by an ICS systems vendor; the requirements developed by the SUIT Working Group (Moran et al., 2021); and the requirements stipulated in IEC 62443-2-3 (IEC, 2015). The vendor wishes to remain anonymous and is thus only referred to as *the vendor*.

3.1 Covered Operational Requirements

We consider the following operational requirements:

R1. The device operator must be in control of the update procedure, i.e., of which update is downloaded for which devices and when it is installed.

R2. The update author must provide meta-data about the released updates, allowing device operators to determine if they are of interest and applicable. The meta-data must inform about which hardware and SW versions the update applies to, and what type of update it is (e.g., configuration files, execute-in-place file, loadable modules).

R3. Different update types from different author types must be supported. Update types include firmware updates, manufacturer libraries, third-party libraries and operator applications. Author types include device manufacturers, device operators and third-party library providers.

R4. The device operator must be able to automate (parts of) the update procedure.

R5. The update procedure must support different update distribution strategies, e.g., push and pull.

3.2 Covered Security Requirements

During the whole process, it must be practically infeasible to: i) access the content and meta-data of SW updates, except for the intended recipients; ii) tamper with SW updates, by altering their content and meta-data; iii) inject bogus or replayed updates, making them appear as valid and legitimate. Consistently, we consider the following security requirements.

R6. Updates and meta-data must be source authenticated and integrity protected. The keys for verification must be available to the device operator.

R7. An adversary must not be able to trick a device into installing updates that are valid and non tampered with, but yet mismatching. These include updates that are obsolete, not intended for the target device, or with unfulfilled preconditions.

R8. The device operator must be able to securely obtain the correct download location for an update.

R9. It must be possible to encrypt the update and its meta-data, to prevent an adversary from performing reverse engineering. The decryption keys must be available to the device operator.

3.3 Non Covered Requirements

The following requirements must also be fulfilled to ensure a secure update solution, but are out of scope for our architecture and need to be addressed separately. Yet, we provide the following guidelines.

• The update process must be reliable and resilient to device malfunctions. For example, a reboot of a device during an update must not lead to a malfunction. This can be achieved by equipping the devices with a second stage bootloader that is capable of performing the updates and resuming them in case of failures.

• The update procedure should be adapted to resource constrained environments. For example, the code-footprint for parsing the meta-data format must be small, and updates that require a device reboot must work with a minimal bootloader. Our approach partly addresses this requirement by offloading the devices through the Update Dispatcher (see Section 4.1).

• If the update requires a reboot, protection is needed against swapping a previously verified update with a malicious one during reboot. This can be addressed by equipping the devices with a secure boot mechanism, that verifies updates during the boot process.

4 UPDATE ARCHITECTURE

Our architecture is shown in Figure 1 and described in Sections 4.1-4.3 fulfills the requirements in Section 3. We describe the update process in Section 5.

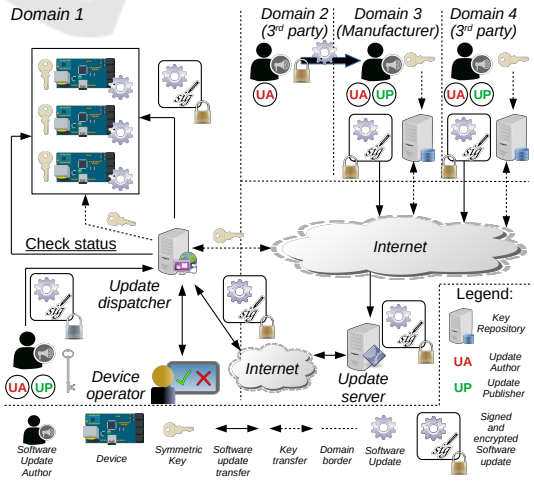

Figure 1: Architecture for secure software update.

4.1 Architecture Entities and Actors

The architecture in Figure 1 includes:
• *Domain.* Networked cyber-physical systems operated by the same administrative authority. For instance, a power plant or a factory.
• *Device.* A networked device operating in a Domain. A Device runs one or many SW components (possibly issued by different sources), belongs to a single Device Operator, and is handled by one Update Dispatcher in the same Domain.
• *Device Manufacturer.* The company that designed and manufactured a Device.
• *Device Operator.* The entity with authority and control of Devices in its Domain for day-to-day operation and maintenance. This includes system/network administrators and authorized maintenance operators.
• *Update Author (UA).* Entity issuing a SW component and related updates. Different Update Authors may release different SW components and updates for a Device. The Update Author may differ from the Update Publisher.
• *Update Publisher (UP).* Entity distributing an update to a previously released SW component. The Update Publisher can be *internal*, i.e., it belongs to the same Domain of the Devices to update and has a secure relationship with the Update Dispatcher of that Domain (see "Domain 1" in Figure 1). Otherwise, the Update Publisher is *external*, i.e., it uploads SW updates to an Update Server (see "Domain 3" and "Domain 4" in Figure 1).
• *Key Repository.* A Server available on the Internet and controlled by a single external Update Publisher, which is in a secure relationship. It stores key material that the associated Update Publisher uses to secure published SW updates.
• *Update Server (US).* A server available on the Internet, where any external Update Publisher uploads SW updates to distribute. Different Update Publishers can use an Update Server, and an Update Publisher can rely on multiple Update Servers for redundancy or scalability.
• *Update Dispatcher (UD).* Domain-local entity that obtains SW updates from Update Servers or internal Update Publishers. The Update Dispatcher is controlled by the Device Operator and is responsible for: i) keeping track of devices in its Domain with regards to the current SW versions and the status of ongoing update processes; ii) interacting with the Device Operator for confirming/revising update operations, and for enforcing fine-grained update policies; iii) distributing updates to the intended devices.
• *Update Key.* A symmetric key generated by an Update Publisher used to encrypt SW updates, which ensures their confidentiality from the Update Publisher to the Domain with Devices to update. An Update Publisher generates a new Update Key upon releasing a new SW update and uploads it to its associated Key Repository.

4.2 Types of SW Updates and Update Authors

Different Update Authors disseminate their SW updates differently based on their expected role and the type of update. We consider three types of SW updates.
• **T1.** SW updates for the *Application Layer*, thus non-critical for the Devices. Any authorized entity can both author and publish these updates.
• **T2.** SW updates for the *Firmware* and *Library* Layer, thus critical for the Devices. They are authored and published only by the Device Manufacturer.
• **T3.** SW updates for the *Firmware* and *Library* Layer, thus critical for the Devices. Unlike for class T2, these updates are *authored* by a trusted 3rd-party developer but are *published* only by the Device Manufacturer after having been approved by it.

Also, we consider three types of Update Authors.
• **A1.** Update Authors in the same Domain of the Devices to update. These Update Authors only produce and distribute SW updates of class T1. They act as internal Update Publishers and directly upload their SW update to the Update Dispatcher over a secure communication channel. Figure 1 shows an Update Author of this class in "Domain 1".
• **A2.** Update Authors that are specifically Device Manufacturers. These Update Authors can produce and distribute SW updates of classes T1 and T2. They act as external Update Publishers by uploading their SW update to the Update Server over a secure communication channel. Figure 1 shows an Update Author of this class in "Domain 3".
• **A3.** Update Authors as authorized 3rd party developers, i.e., non Device Manufacturers. These Update Authors can produce and distribute SW updates of classes T1 and T3. For a class T1 update, the Update Authors act as the Update Publisher and upload the SW update to the Update Server over a secure communication channel. Figure 1 shows an Update Author of this class as releasing a SW update of class T1 in "Domain 4". For a class T3 update, these Update Authors do *not* act as Update Publisher, but rather provide their SW update to the Device Manufacturer over a secure communication channel. The Device Manufacturer verifies the SW update and uploads the SW update to the Update Server, acting as an external Update Publisher on behalf of the Update Author.

Table 1: Division of roles Update Author and Update Publisher for types of SW Updates (Author: "Update Author"; Publisher: "Update Publisher").

Author	SW Update type		
	T1	**T2**	**T3**
A1	Author & Publisher		
A2	Author & Publisher	Author & Publisher	Publisher
A3	Author & Publisher		Author

Figure 1 shows an Update Author of this class in "Domain 2", as providing its SW update to the Device Manufacturer in "Domain 3" for approval and following upload to an Update Server.

Table 1 overviews the types of Update Authors in relation to the different classes of SW updates.

4.3 Security Model

We assume the distribution process is secured against an active, on-path adversary. The adversary can capture, delete, modify, and change the order of all messages between involved parties. The adversary can also inject previously captured or newly crafted messages. However, the adversary does not own the required key material to participate in such communications and cannot break the used cryptographic functions.

The entities involved in the update distribution protect communications to ensure confidentiality, freshness, integrity and source authentication of messages. Communications to be secured are among: i) a third-party developer and a Device Manufacturer; ii) an external Update Author and an Update Server; iii) an Update Server and an Update Dispatcher; iv) an Update Dispatcher and the Key Repository of an external Update Publisher; v) an Update Dispatcher and the Device Operator; vi) an Update Dispatcher and an internal Update Author; vii) an Update Dispatcher and the Devices to update. As long as the requirements above are fulfilled, our approach is not limited to a specific secure communication protocol.

The security of entities such as the Key Repository, the Update Server, and the Update Dispatcher is also critical for the security of the distribution process. Those entities are expected to be equipped with plenty of resources as well as properly implemented and managed to be robust, reliable, and secure, thus practically infeasible to compromise. To this end, best engineering practices exist (Birman, 2012).

Accessing the Key Repository, Update Server, and Update Dispatcher must be limited to authorized requesters and only for performing authorized operations. Thus, secure and fine-grained access control must be enforced.

In particular, an Update Publisher has to allow Update Dispatchers to access its Key Repository to retrieve Update Keys when needed. This requires ensuring fine-grained access control at the Key Repository. To this end, a well-established approach is the OAuth 2.0 Authorization framework (Hardt, 2012), with the Update Publisher, the Key Repository, and the Update Dispatchers acting as Resource Owner, Resource Server, and Clients, respectively.

A Device Operator has to allow Update Dispatchers to access their devices to distribute the SW updates. In order to enforce fine-grained access control, a possible, well-established approach is the ACE framework for Authentication and Authorization in Constrained Environments (Seitz et al., 2022), with the Device Operator, the Devices, and the Update Dispatcher acting as Resource Owner, Resource Servers and Client, respectively.

5 UPDATE PROCESS

With reference to Figure 1, the SW update process occurs as per the three-phase workflow below.

5.1 Update Release

The Update Author generates a new version of the SW to be updated, as an incremental patch to a previous existing version, or a full-fledged updated version. If the SW update is of type T3 (see Section 4.2), the Update Author, of type A3, first provides the SW update to the Device Manufacturer acting as external Update Publisher. This relies on a secure communication channel, ensuring message confidentiality, integrity and source authenticity. The Device Manufacturer decrypts the SW update from the Update Author, and verifies it to be fine to distribute. If so, the Device Manufacturer acts as external Update Publisher as described below.

For each Update Author type, confidentiality of the SW update is ensured. In fact, an Update Author of type A1 (see Section 4.2) acts as internal Update Publisher, and provides a SW update to the Update Dispatcher, over a secure communication channel.

Instead, if the Update Author is of type A2 or A3, the external Update Publisher encrypts the SW update, before publicly releasing it. This is not trivial to achieve, as the Update Publisher wishes to ensure confidentiality of the SW update all the way to the Device domain. That is, the content of the SW update should be accessible to the Devices in "Domain 1", while it should remain opaque to (untrusted) intermediaries in other domains, e.g., the Update Server. This can be achieved by encrypting the SW update

with a symmetric key shared among the Update Publisher and the target Devices. However, the Update Publisher is not expected to know what Devices are targeted by the released SW update. This prevents relying on a (poorly scalable) establishment of symmetric keys with the target Devices.

To overcome this issue, the Update Publisher first generates a symmetric key *K* associated with this update release, and a corresponding key identifier *kid*. The Update Publisher uses *K* to encrypt the SW update, and stores *K* as well as *kid* in the Key Repository under its control and in its same domain. Then, the Update Publisher signs the encrypted SW update together with *kid*, by using its private key. Finally, the Update Publisher publicly releases the signed and encrypted SW update, by uploading it to the Update Server. As the Update Publisher used the *encrypt-then-sign* construction, the Update Server can authenticate and verify the integrity of the SW update, by using the public key of the Update Publisher to verify its signature.

5.2 Update Fetching

The Update Dispatcher has a secure association with the Update Publisher. Also, if the Update Publisher is *external*, the Update Dispatcher has fine-grained secure access to the associated Key Repository.

Next, the Update Dispatcher collects SW updates for Devices in its same Domain, by sending PULL requests to, or receiving PUSH notifications from, the entity hosting updates of interest. This can be an *internal* Update Publisher (see Figure 1, "Domain 1"), or an Update Server where *external* Update Publishers have uploaded an update (see Figure 1, "Domain 3" and "Domain 4"). In either case, the Update Dispatcher can authenticate and verify the integrity of the update, using the public key of the Update Publisher to verify its signature. For updates of type T2 or T3 (see Section 4.2), the Update Dispatcher verifies that they are signed by the Device Manufacturer.

Then, the Update Dispatcher retrieves the key *K*, possibly from the Key Repository associated with the *external* Update Publisher, by using the *kid* specified in the update. Then, the Update Dispatcher decrypts the update using *K*, accesses its content, and determines the next steps to take for the actual distribution to the target Devices, possibly in concertation with the Device Operator (see Section 5.3). Note that, thanks to the digital signature from the Update Publisher, the Update Dispatcher is not able to alter the content of the update.

Finally, the Update Dispatcher provides *K* to the target Devices, over an associated secure communication channel.

5.3 Update Distribution

The Update Dispatcher does *not* necessarily distribute the SW update to the Devices right after having retrieved it. That is, the update distribution is scheduled as follows.

First, the Update Dispatcher determines which Devices to update, e.g., based on which SW version they are running.

Second, the Update Dispatcher enforces update policies on behalf of the Device Operator, to reflect preferences, requirements and priorities in performing SW updates. For instance, it may be fine to immediately distribute non-critical updates, if coming from third-party providers of utility SW libraries. Instead, some updates may have to be explicitly approved and/or postponed to not interfere with safety-critical operations in the networked system. This requires the Update Dispatcher to "put on hold" the update process, and interact with the Device Operator for a final approval and/or possible rescheduling. Note that, while feasible for a unit such as the Update Dispatcher, it is not realistic to expect constrained and intermittently available Devices to effectively and promptly interact with the Device Operator by themselves.

In due time and according to the policies and interaction above, the Update Dispatcher distributes the SW update to the Devices over a secure communication channel.

Then the Devices receiving the update: i) authenticate and check the integrity of the update, by using the public key of the Update Publisher to verify its signature; ii) decrypt the update by using the key *K* originally used by the Update Publisher, as received from the Update Dispatcher (see Section 5.2); and iii) install the SW components from the received update according to vendor- or author-specific procedures, which are out of scope for this paper.

6 SECURITY ANALYSIS

This section discusses how relevant security requirements are met and supporting security services.

6.1 Integrity and Source Authentication of SW Updates

We achieve these requirements by signing the encrypted SW update using the Update Publisher pri-

vate key. A Device can ensure that a received update is not tampered with and comes from the intended Update Publisher. *Encrypt-then-sign* allows (untrusted) intermediaries to verify the validity of an update during distribution. Intermediaries include the Update Server, the Update Dispatcher, and border routers used as the entry point to local networks. Public keys of Update Publishers can be pre-provisioned or retrieved from a trusted key repository such as a PKI.

For SW updates of type T3 (see Section 4.2), it may be convenient that the Update Author, of type A3, instead uses a construction *sign-then-encrypt* when providing the update to the Device Manufacturer (see Section 5.1). In fact, after decrypting the received update, the Device Manufacturer can preserve the original signature from the Update Author before re-encrypting the update for upload to the Update Server. The advantage is that, later on, both the Update Dispatcher and the Devices to update have access to the original signature from the Update Author. Thus, they can also verify that: i) the update comes from that Update Author, despite the Device Manufacturer acting as an external Update Publisher, and ii) the Device Manufacturer has not altered the original update but only approved it as is before distribution. This comes at the cost of handling the public keys of such Update Authors and verifying the signature from one such Update Author. Public keys of these Update Authors can be pre-provisioned or retrieved from a trusted key repository.

6.2 Confidentiality of SW Updates

Unlike integrity and source authentication, this requirement is often optional to fulfill. However, it allows for hiding update contents from unintended recipients and impeding reverse engineering of the distributed SW. This prevents reverse engineering of updates for vulnerability discovery or intellectual property infringement. Devices in physically insecure environments can further leverage secure processing and storage of received updates.

Our approach ensures that, in the target Domain, only the intended recipient Devices and the supporting Update Dispatcher can access the update's contents using the key K that only authorized actors can retrieve from the Key Repository of the Update Publisher. Also, the content of distributed updates remains inaccessible to intermediaries between the Update Publisher and the Update Dispatcher. These include the Update Server, which cannot access the stored updates and is not a valuable target for an attacker trying to take control of gaining access to stored updates.

The above requires that Devices trust their domain Update Dispatcher, as the point where end-to-end confidentiality with the Update Publisher ends, due to the Update Dispatcher fetching K from the Key Repository of the Update Publisher. Based on the update's content, the Update Dispatcher takes the correct next steps (e.g., interacting with the Device Operator), but it cannot alter the update's content due to the Update Publisher's digital signature.

6.3 Supporting Security Services

The open standard CBOR Object Signing and Encryption (COSE) (Schaad, 2017) can be used to efficiently encrypt, sign and compute a Message Authentication Code (MAC) on the SW updates achieving small messages that can be efficiently processed.

Open frameworks for authentication and authorization, such as OAuth 2.0 (Hardt, 2012), and ACE (Seitz et al., 2022), can be used to enforce fine-grained access control. This concerns Update Dispatchers and external Update Publishers accessing Key Repositories or Update Servers; and the Update Dispatcher accessing the Devices to update. In the latter case, the ACE framework is preferable, as it is designed to enforce access control on resource-constrained devices.

6.4 Secure Communication

As mentioned in Section 4.3, the message exchanges are secured against active on-path adversaries.

To this end, the involved parties use secure communication protocols to protect message delivery. Especially for securing traffic over the Internet, available protocols include: i) the TLS (Rescorla, 2018) and DTLS (Rescorla et al., 2022) suites providing security at the transport layer, and for which related profiles targeting resource-constrained devices have been defined (Tschofenig and Fossati, 2016); and ii) the OSCORE protocol (Selander et al., 2019), which provides security for CoAP messages at the application layer through COSE, and is explicitly designed to suit resource-constrained devices.

Alternatives can be implemented at the network or data-link layer, e.g., in the domain hosting the target Devices to update, where ad-hoc (legacy) communication protocols may be used. As long as source-authentication, integrity, confidentiality, and freshness of exchanged messages are ensured, our approach is not devoted to a particular security protocol, and leaves the choice to the application and deployment.

7 CONCLUSION

We have presented a novel approach for securely distributing SW updates over the Internet, to devices in Industrial Control Systems (ICS). Our approach addresses several requirements provided by a real-world ICS operator, and has the following benefits: it enables the secure distribution of different types of SW updates, originated by different types of authors; it leaves the device operator in control of the update process, even for third-party SW updates; it ensures not only integrity and source authentication, but also confidentiality of SW updates. Furthermore our approach builds on well established and adopted foundations such as fine-grained authentication and secure sessions. To the best of our knowledge, this is the first approach for secure SW distribution that has all these benefits. Future work will focus on the implementation and performance evaluation of a SW update process based on our approach, considering real ICS platforms as devices to update.

ACKNOWLEDGMENTS

This work was supported by the SSF project Sec4Factory (grant RIT17-0032); by VINNOVA through the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652). The authors would like to thank Arash Vahidi for the helpful discussions and feedback.

REFERENCES

Ambrosin, M., Busold, C., Conti, M., Sadeghi, A.-R., and Schunter, M. (2014). Updaticator: Updating billions of devices by an efficient, scalable and secure software update distribution over untrusted cache-enabled networks. In *European Symposium on Research in Computer Security*, pages 76–93. Springer.

Arakadakis, K., Charalampidis, P., Makrogiannakis, A., and Fragkiadakis, A. (2021). Firmware over-the-air programming techniques for iot networks-a survey. *ACM Computing Surveys (CSUR)*, 54(9):1–36.

Asokan, N., Nyman, T., Rattanavipanon, N., Sadeghi, A.-R., and Tsudik, G. (2018). ASSURED: Architecture for Secure Software Update of Realistic Embedded Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2290–2300.

Birman, K. P. (2012). *Guide to Reliable Distributed Systems. Building High-Assurance Applications and Cloud-Hosted Services*. Springer.

Catuogno, L., Galdi, C., and Persiano, G. (2017). Secure dependency enforcement in package management systems. *IEEE Transactions on Dependable and Secure Computing*, 17(2):377–390.

Davidson, C. C., Andel, T., Yampolskiy, M., McDonald, J. T., Glisson, B., and Thomas, T. (2018). On SCADA PLC and Fieldbus Cyber-Security. In *The International Conference on Cyber Warfare and Security*. Academic Conferences and Publishing International.

El Jaouhari, S. and Bouvet, E. (2022). Secure firmware over-the-air updates for iot: Survey, challenges, and discussions. *Internet of Things*, 18:100508.

Fassino, J. (2016). Secure Firmware Update in Schneider Electric IOT-enabled offers. In *Internet of Things Software Update Workshop (IoTSU)*. Internet Architecture Board (IAB).

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard). Updated by RFC 8252.

Hernández-Ramos, J. L., Baldini, G., Matheu, S. N., and Skarmeta, A. (2020). Updating iot devices: challenges and potential approaches. In *2020 Global Internet of Things Summit (GIoTS)*, pages 1–5. IEEE.

IEC (2015). Security for industrial automation and control systems – Part 2-3: Patch management in the IACS environment. Technical Report IEC TR 62443-2-3, International Electrotechnical Commission (IEC).

IETF (2022). SUIT - Software Updates for the Internet of Things.

Moran, B., Tschofenig, H., and Birkholz, H. (2022). A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices. RFC 9124.

Moran, B., Tschofenig, H., Brown, D., and Meriac, M. (2021). A firmware update architecture for internet of things. RFC 9019.

Mugarza, I., Flores, J. L., and Montero, J. L. (2020). Security issues and software updates management in the industrial internet of things (iiot) era. *Sensors*, 20(24):7160.

Petrov, S. (2018). Patch Delivery Infrastructure in SCADA Systems. Master's thesis, KTH Royal School of Technology.

Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446.

Rescorla, E., Tschofenig, H., and Modadugu, N. (2022). The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. RFC 9147.

Schaad, J. (2017). CBOR Object Signing and Encryption (COSE). RFC 8152 (Proposed Standard).

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and Tschofenig, H. (2022). Authentication and authorization for constrained environments using the oauth 2.0 framework (ace-oauth). RFC 9200.

Selander, G., Mattsson, J. P., Palombini, F., and Seitz, L. (2019). Object security for constrained restful environments (oscore). RFC 8613.

Tschofenig, H. and Fossati, T. (2016). Transport layer security (tls) / datagram transport layer security (dtls) profiles for the internet of things. RFC 7925 (Proposed Standard).