Transparent End-to-End Security for Publish/Subscribe Communication in Cyber-Physical Systems

Markus Dahlmanns*, Jan Pennekamp*, Ina Berenice Fink*,
Bernd Schoolmann*, Klaus Wehrle*, Martin Henze‡
*Communication and Distributed Systems, RWTH Aachen University, Aachen, Germany
‡Cyber Analysis & Defense, Fraunhofer FKIE, Wachtberg, Germany
{dahlmanns, pennekamp, fink, schoolmann, wehrle}@comsys.rwth-aachen.de · martin.henze@fkie.fraunhofer.de

ABSTRACT

The ongoing digitization of industrial manufacturing leads to a decisive change in industrial communication paradigms. Moving from traditional one-to-one to many-to-many communication, publish/subscribe systems promise a more dynamic and efficient exchange of data. However, the resulting significantly more complex communication relationships render traditional end-to-end security futile for sufficiently protecting the sensitive and safety-critical data transmitted in industrial systems. Most notably, the central message brokers inherent in publish/subscribe systems introduce a designated weak spot for security as they can access all communication messages. To address this issue, we propose ENTRUST, a novel solution for k**e**y server-based e**n**d-**t**o-end secu**r**ity in p**u**blish/**s**ubscribe sy**s**tems. ENTRUST transparently realizes confidentiality, integrity, and authentication for publish/subscribe systems without any modification of the underlying protocol. We exemplarily implement ENTRUST on top of MQTT, the de-facto standard for machine-to-machine communication, showing that ENTRUST can integrate seamlessly into existing publish/subscribe systems.

CCS CONCEPTS

• **Security and privacy → Security protocols**; • **Networks → Application layer protocols**; • **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

cyber-physical system security, publish-subscribe security, end-to-end security

ACM Reference Format:
Markus Dahlmanns, Jan Pennekamp, Ina Berenice Fink, Bernd Schoolmann, Klaus Wehrle, Martin Henze. 2021. Transparent End-to-End Security for Publish/Subscribe Communication in Cyber-Physical Systems. In *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-physical Systems (SAT-CPS'21), April 28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3445969.3450423

SAT-CPS'21, April 28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8319-6/21/04…$15.00
https://doi.org/10.1145/3445969.3450423

1 INTRODUCTION

Historically, production and manufacturing deployments are realized in separated networks and based on one-to-one communication, i.e., one device requests the state of another device to adapt its processes. For example, SCADA systems periodically request the state of connected programmable logic controllers (PLCs) or remote terminal units (RTUs) and respond with control commands or updates of setpoints [15, 16].

In modern deployments, this antiquated simplistic communication pattern will change drastically: Movements such as the Industry 4.0 [30] and the Internet of Production [42] will deploy more and more sensors to provide data for future production or processing steps [20] as well as further analysis [41]. As a result, a vast exchange of (sensor and control) data will emerge, leading to dynamic communication relationships between various endpoints [43] and moving from one-to-one to many-to-many communication [29, 46].

The most promising communication paradigm to address these ascending communication demands is Publish/Subscribe (PubSub), which significantly reduces the complexity of many-to-many communication systems by decoupling senders and recipients using a central message broker [14]. However, while transmitted data is often extremely sensitive [15, 41], and *secure* communication is inevitable to meet confidentiality and safety requirements, PubSub complicates the establishment of secure connections. Contrary to one-to-one communication, where traditional end-to-end (E2E) secure protocols, e.g., basing on TLS or the OPC UA Binary Protocol, can be used (although many industrial deployments fail to do so correctly or at all [6]), many-to-many communication requires more than two partners to negotiate key material, making established handshake procedures impossible. As a result, the message broker as the central component constitutes a precarious weak spot for security in PubSub-based industrial communication.

Indeed, related work predicts more attacks against CPSs in future [10] and has shown that many Internet-facing message brokers are configured insecurely [32] enabling attackers to eavesdrop on sensitive information or inject commands into communication. However, even correctly secured brokers might violate the security and privacy of involved stakeholders. Thus, apart from offering a plethora of suitable targets, the brokers need to be fully trusted not to compromise the integrity and confidentiality of transmitted data.

In this paper, we address the pressing issue of missing E2E security in PubSub communication, which requires unrealistically high trust in centrally managed message brokers. To this end, we propose ENTRUST, a novel and efficient scheme for k**e**y server-based e**n**d-**t**o-end secu**r**ity in p**u**blish/**s**ubscribe sy**s**tems. ENTRUST

enables confidential, integrity protected, and authenticated communication utilizing the PubSub paradigm without any trust requirements regarding on-path message brokers. Notably, ENTRUST seamlessly integrates into existing PubSub protocols. It neither requires changes to the underlying PubSub protocol nor the message broker and does not introduce any additional Internet-reachable entities, allowing for easy deployment without increasing the attack surface. Thus, ENTRUST can be gradually deployed in any existing CPS scenario relying on PubSub communication as long as all participants in a certain message flow support ENTRUST.

While ENTRUST is not first to focus on security in PubSub systems, previous works are either difficult to deploy, e.g., by requiring changes to the broker implementation [5, 26, 49] or require to expose new services, thus introducing new attack vectors [29, 37].

Contributions. Our main contributions are as follows.

- We distill the benefits of many-to-many communication in CPS and identify the need for E2E security to protect against misconfigured, insecure, or malicious message brokers.
- We propose ENTRUST, a novel and transparent E2E security protocol for well-established PubSub communication. ENTRUST exclusively relies on already enabled Internet-facing services and thus does not require any changes to the PubSub protocol or any (on-path) message broker.
- We show the feasibility and applicability of ENTRUST on top of the predominantly used PubSub protocol MQTT.

Organization. After introducing real-world many-to-many communication scenarios and the benefits of the PubSub paradigm in Section 2, we derive the need for E2E security in Section 3. In Section 4 we show that related work is not able to cover all of our identified needs, which is why we propose ENTRUST, our protocol allowing for transparent and secure communication in PubSub schemes in Section 5. Afterward, we discuss ENTRUST's security properties in Section 6 and further demonstrate its (computational) feasibility in Section 7. Finally, Section 8 concludes this paper.

2 MANY-TO-MANY COMMUNICATION IN CYBER-PHYSICAL SYSTEMS

In traditional, isolated production networks, devices communicate in a one-to-one fashion using proprietary protocols [11, 48], such as Modbus [35] or ProfiNet [25]. These protocols mainly allow requesting specific values other devices hold and respond with control commands or value updates, e.g., as in SCADA systems.

However, Industry 4.0 [30] drastically changes communication patterns and demands, as industrial networks (i) grow with more and more systems generating and receiving data [17, 46], (ii) require unified protocols to enable this variety of devices to communicate with each other [2], as well as (iii) move closer to other computer networks and the Internet [4, 6, 48], all to (iv) enable more and more different entities to receive production data for different use cases [18], e.g., production-to-production communication [43].

To address these requirements of future industrial communication, where an increasing number of devices generate data, and more and more entities require to receive this data, i.e., a shift to many-to-many machine-to-machine communication occurs, Publish/Subscribe (PubSub) communication significantly reduces complexity [14]. While traditional server/client communication requires entities to maintain a state for every participant, with PubSub, every participant has to maintain a single state only. To this end, the PubSub paradigm introduces a central broker, to which all communication partners establish a connection, and thus decouples senders and receivers. Instead of maintaining multiple one-to-one connections, senders now simply publish their data to the central broker, which relays the data to all receivers that subscribed to a specific content (in content-based PubSub) or topic (in topic-based PubSub). Promising protocols for industrial communication, such as OPC UA, already provide support for PubSub communication [37], highlighting its relevance.

While the introduced broker reduces the connection complexity, it also presents a critical element regarding security as it is used to transmit the entirety of data CPSs generate. This data encompasses information such as readings [32], which might allow conclusions on the current production process, e.g., which products are produced or how well a specific factory performs [44], and commands controlling specific CPSs [32], e.g., movement commands for robots.

Takeaway: *PubSub communication is a promising candidate to account for the needs of industrial (machine-to-machine) dataflows and reduce the complexity in the face of a large number of new many-to-many patterns. However, the added central broker challenges the security of all passed messages as it mediates and relays all dataflows.*

3 THE NEED FOR E2E SECURITY

While the central broker instance processing all communication allows to reduce the communication complexity, it also introduces a sweet spot for attackers as it is in control over all (safety-critical) control messages and (potentially confidential) sensor data.

To achieve their privacy and safety regulations, i.e., keep their transmitted data private and prevent unauthorized clients from sending commands, operators face two challenges when setting up a broker. First, operators must ensure that only authorized clients are able to send and receive data via the communication infrastructure, and second, no instance on the communication path should be able to inject, alter, or drop data. Both goals do not only encompass clients connected to a broker but also the broker itself.

Below, we identify two vectors for attackers to gain access to this sweet spot (Section 3.1) and subsequently derive requirements for E2E security to eliminate these vectors (Section 3.2).

3.1 Attack Vectors in PubSub Communication

The security of PubSub communication, i.e., confidentiality and authenticity of messages, is strongly tied to the security of the broker mediating all dataflows. As shown in Figure 1, this situation results in two primary attack vectors on PubSub communication: misconfigured brokers as well as malicious or compromised brokers.

Misconfigured Broker: The simplified communication structure introduced by PubSub also simplifies attacks. While it is straightforward for publishers and subscribers to connect to a broker and exchange data via specific topics, it is also easy for attackers to do so, i.e., connect to a broker as well as publish and/or subscribe to specific topics. Consequently, an attacker could inject commands into ongoing communication within a specific topic, directly violating safety requirements, as well as receive a copy of all messages for a specific topic, breaking confidentiality demands.

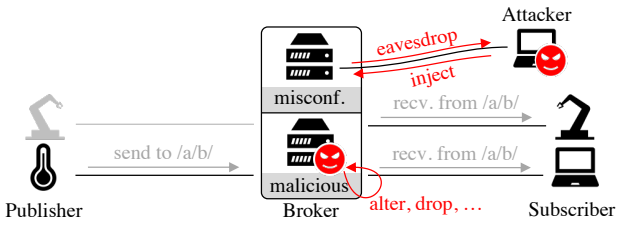

Figure 1: Twofold risks in publish/subscribe deployments; misconfigured or malicious message brokers put secure communication at risk as a sweetspot for attackers.

To address these security issues, many PubSub protocols provide authentication mechanisms [3, 19], allowing brokers to restrict access to specific topics to authenticated and authorized clients. However, recent research has shown that many Internet-reachable brokers implementing MQTT [3], one of the predominantly used PubSub protocols, do not implement access control correctly [32]. These brokers allow everybody on the Internet to receive the transmitted data and potentially inject messages as no further protection is applied to the communication. This undesirable state highlights the need to protect against misconfigured brokers, as any configuration flaw directly impacts security.

Malicious or Compromised Broker: Even if fully secure implementation and configuration of brokers could be assumed, as of now, full trust in the broker is required, as brokers (i) have to implement and enforce configured access rules as instructed and (ii) must not eavesdrop or alter any transmitted messages. In the case of a malicious or compromised broker, these trust assumptions do not hold, and both confidentiality and safety are at stake. Hence, a malicious broker and corresponding attackers would be able to eavesdrop on every transmitted message to conclude about (production) processes the CPSs are involved in, or even alter, inject or drop messages to control and disturb these processes.

The risk of malicious or compromised brokers further increases when operators do not set up their own broker infrastructure but instead rely on cloud infrastructure or services, where (often unknown) third-party entities are responsible for correctly setting up and maintaining the broker [21, 22, 26]. Consequently, not only configuration and implementation errors but also malicious intentions or compromises of brokers (potentially run on third-party infrastructure) severely threaten the confidentiality and safety of CPSs communicating globally over PubSub protocols.

3.2 Requirements for E2E Security

The main problem leading to both attack vectors results from an inherent design decision of PubSub communication: To efficiently distribute data received from publishers to several subscribers, the broker is given access to all transmitted data in plaintext [5]. To solve this issue and thus take the burden of being a valuable target for attacker off the broker, we are strongly convinced that it is imperative to protect transmitted payload against changes and access by unauthorized entities between sender and receiver using E2E security [50]. Accounting for the in current PubSub protocols unconsidered support for E2E security, we identify five essential requirements that must be addressed to improve the state of modern PubSub systems and mitigate the existing attack vectors.

R1: Information Security. To prevent that any entities other than the legitimate subscribers are able to read or alter the communication or intercept the connection, the E2E security scheme must ensure confidentiality, integrity, and authenticity [39, 53]. While confidentiality is required to keep valuable information such as process parameters private, integrity protection is crucial to prevent external entities from altering messages containing commands to control machines, as these could potentially cause production outages or harm humans. Furthermore, authenticity allows subscribers to verify that messages are sent by a legitimate publisher, i.e., unauthorized entities are not able to undetectably inject messages into ongoing communication.

R2: Detection of Dropping and Duplication. In addition to ensuring that transmitted data is not altered and no additional data is injected into an ongoing communication, it is also important to enable subscribers to detect dropping or duplication of legitimate communication messages [39]. Dropped or duplicated and replayed messages could interfere with ongoing production processes leading to commands for specific actions being not performed or being performed more than once. Both aspects introduce a significant safety problem [7, 21], i.e., outages or harm to humans.

R3: Decoupling of Senders and Receivers. An important property of PubSub is the decoupling of senders and receivers [5, 29], allowing publishers to send messages without any knowledge on the subscribers retrieving these messages. Consequently, E2E security has to preserve this foundational property of PubSub, as otherwise communication complexity would again be increased, superseding the benefits of PubSub.

R4: Deployability. To ensure wide use of E2E security, it has to be designed in a deployable manner, i.e., security mechanisms must not introduce fundamental changes to current PubSub deployments. Here, it is especially important that the broker implementation remains untouched. Changes to the broker would prevent operators from using E2E security whenever it is maintained by a third party, e.g., a cloud provider. Furthermore, the scheme must not require any services other than the broker being reachable directly for the clients as any addition could increase the attack surface, especially when a service is Internet-facing. Likewise, any system should support a reasonable level of granularity in terms of security policies and access control to account for the varying needs of individual dataflows in PubSub communication [29].

R5: Efficiency. Finally, it is essential that E2E security is efficient by not introducing significant communication latency and data overhead. As E2E security introduces more load on the end hosts, which in many CPS applications have only limited resources [29], it is important that all security operations can be performed efficiently, as complex mechanisms would introduce too much latency. Furthermore, applied security mechanisms must not add too many additional data to be transmitted, which would require larger capacity in industrial networks. Especially in PubSub deployments, where messages are distributed to various recipients, even small overhead leads to a significant added burden.

Takeaway: *The central broker in PubSub is an attractive target for attackers. E2E security allows to take this burden off the broker, but any attempt to realize it has to consider five distinct design requirements.*

Approach	Information Security (R1)	Dropping / Duplication Detection (R2)	Decoupling (R3)	Deployability (R4)	Efficiency (R5)
Naïve, e.g., [9, 24, 45]	●	◑	○	●	○
FRAMEWORK [39]	●	◑	●	◑	◑
SMQTT [50]	◕	○	●	◕	◔
PICADOR [5]	◕	○	●	◑	◔
OPC UA [37]	●	●	●	◔	●
JEDI [29]	●	○	●	◔	◑
MQT-TZ [49]	◑	●	●	◑	◑
MOUCON [26]	◔	○	●	◑	◕

Table 1: None of the related work approaches covers all requirements for secure PubSub communication in industrial environments. Thus, to mitigate any unsafe conditions, the E2E security of such infrastructures should be revisited.

4 RELATED WORK

While the security of PubSub systems has been studied extensively, e.g., [33, 51], realizing E2E security for PubSub communication has only been studied by few approaches to date. In this section, we survey approaches to realize E2E security for PubSub communication from related work and analyze, whether they meet our generic requirements for E2E security in PubSub systems (cf. Section 3.2). We provide an overview on our survey of related work in Table 1.

Naïve Approach: Outside of PubSub communication, E2E security is a valuable concept to thwart different attacks, e.g., eavesdropping or modification attacks. Most notably, (D)TLS [9, 45] allows to realize E2E security for Internet communication and specifically tailored variants for resource-constrained CPSs exist, e.g., [24].

While a naïve approach to realize E2E security would be to apply these already established mechanisms on top of the PubSub protocol, such an approach cannot meet the requirements that we identified in Section 3.2: Although these protocols implement information security (R1), would be deployable without any changes to the broker when applied on top of the PubSub protocol (R4), and typically implement duplication and dropping detection using nonces (R2), this approach would completely destroy all benefits introduced by PubSub. These protocols would require the publishers to perform a handshake with all subscribers, to encrypt a message for every subscriber independently giving up the sender/receiver decoupling (R3), and introducing a tremendous overhead (R5).

FRAMEWORK: In the context of the Narada Brokering system [38], Pallickara et al. [39] propose a framework that allows clients to communicate via secure topics where the communication is E2E protected. To this end, on behalf of a topic owner, key management centers manage encryption keys as well as access tokens and send them to authorized clients. The key management centers only provide the key material and access tokens to authorized clients by checking their client certificate, which prevents other (malicious) entities from eavesdropping and modifying messages (R1). Furthermore, in the design, senders and receivers are still decoupled (R3).

However, while publishers include a timestamp in the message that enables subscribers to detect duplicated messages, subscribers are not able to detect dropped messages in ongoing communication (R2). Furthermore, FRAMEWORK relies on the functionality of the Narada Brokering system and adapts the broker to enforce access regulations, i.e., it cannot be used on already deployed infrastructure with well established PubSub protocols (R4). Regarding authentication, FRAMEWORK relies on a public key infrastructure, which could be considered inefficient on constrained devices (R5).

SMQTT: The approach of Singh et al. [50] achieves E2E security by utilizing attributed-based encryption (ABE). Here, the broker (or a separate key generation system) distributes key secrets to publishers and subscribers depending on their owned ABE attributes. Thus, the decoupling of senders and receivers is supported by design (R3).

While it would violate the requirement of information security when the broker is able to access and alter transmitted messages, with a separate instance for key management, SMQTT is able to prevent that malicious brokers modify or eavesdrop messages (R1). This implementation decision also affects the requirement of deployability. Already established brokers could be used without any changes if a separate key management instance is used. However, this change complicates the corresponding communication with involved clients (R4). So far, SMQTT does not consider any countermeasures against dropped or duplicated messages (R2) and the use of ABE limits its deployability on constrained CPS devices (R5).

PICADOR: Borcea et al. [5] propose a design, where proxy re-encryption enables the broker to obliviously re-encrypt messages received from the publishers for every subscriber independently, i.e., without the broker itself being able to decrypt a message.

This scheme is able to realize sender/receiver decoupling (R3) and by introducing on-demand secure side channels (realized using a trusted third party) it targets to deploy required key material without the need of an additionally reachable service (R4). However, it requires changes to the broker, thus conflicting our deployability requirement (R4) and adds a massive overhead on the broker which has to re-encrypt each message for every connected subscriber (R5). Furthermore, it does not provide any mechanisms to detect dropping or duplication of messages (R2) and does not guarantee message integrity (R1).

OPC UA PubSub: The PubSub protocol of OPC UA [37] introduces a trusted third party that manages security keys to achieve confidentiality on a per-group level. Operators have to assign publishers and subscribers to groups to enable them to retrieve keys.

OPC UA PubSub fulfils most of our requirements, i.e., applying encryption and signatures to messages for sufficient information security (R1), including a sequence number in the message header and signature for dropping and duplication detection (R2), as well as not introducing significant overhead (R5). Furthermore, it preserves the decoupling of senders and receivers as a prime characteristic of PubSub (R3). However, the introduced trusted third party must be

reachable out-of-band for all publishers and subscribers regularly to renew key material. Hence, it potentially introduces a large variety of new attack vectors (R4).

JEDI: JEDI [29] relies on identity-based encryption to implement E2E information security (R1), while retaining the decoupling of senders and receivers in PubSub communication (R3).

However, JEDI does not enable the receivers to detect dropped or duplicated messages as no nonce or counter value is part of the transmitted packet (R2). Furthermore, similar to OPC UA PubSub, it introduces a trusted third party that derives security keys for a specific topic and time frame. This third party needs to be reachable for the publishers and receivers out-of-band, potentially introducing new attack vectors (R4). Lastly, while relying on the lightweight AES-CTR mode for the symmetric encryption itself, the AES key for each message has to be decrypted via identity based encryption adding even more overhead (R5).

MQT-TZ: Segarra et al. [49] propose to tackle the problem of malicious or compromised brokers by confining the broker software in a trusted execution environment [31]. While not disrupting the sender/receiver decoupling (R3), this approach allows to attest that the broker software is executed as intended, i.e., TLS for the connection between the broker and the publishers and subscribers is correctly implemented and the broker does not alter, drop, or duplicate any messages (R2). This approach is especially promising when brokers are outsourced to the cloud [26].

However, since TLS does not provide E2E security between authorized publishers and subscribers, MQT-TZ does not prevent malicious clients that were able to join the PubSub network to eavesdrop messages of or inject messages into ongoing communication (R1). Furthermore, trusted execution environments are not available on every server, i.e., being difficult to deploy (R4), and introduce a significant overhead on the broker (R5). Additionally, several attacks on trusted execution environments were shown [47] rendering its benefit questionable.

MOUCON: Following orthogonal research, Jia et al. [26] identified different issues in access control capabilities of MQTT brokers operated in the cloud and proposed MOUCON, which checks the access rights to a message for every client based on specific policies.

While this approach holds up sender/receiver decoupling (R3) and is comparatively efficient by only introducing a small overhead on the broker for policy checks (R5), it requires changes to the broker, complicating deployability (R4). Furthermore, with MOUCON applied, still full trust into the broker is required, i.e., the broker is still able to drop or duplicate packets (R2) as well as read or change transmitted packets and even inject new messages (R1).

Takeaway: *Current approaches for E2E security for PubSub cannot satisfy all requirements necessary in industrial environments at once. Related work does not sufficiently solve important requirements such as information security or deployability. Especially w.r.t. safety requirements in and the increasing interconnection of CPSs, an approach covering all of the identified requirements is needed.*

5 ENTRUST — E2E SECURITY FOR PUBSUB

None of the currently available protocols meets all requirements for secure PubSub communication in industrial infrastructures (cf. Section 3.2). Hence, we set out to develop ENTRUST, our protocol allowing **k**ey server-based e**n**d-**t**o-end secu**r**ity in p**u**blish/**s**ubscribe sys**t**ems. The core idea of ENTRUST is to establish a symmetric key per topic (or message channel) which can be used to secure exchanged messages. This topic-specific key is then shared with all authorized publishers and subscribes of the corresponding topic.

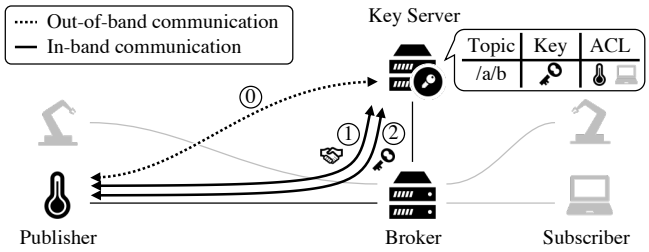

Figure 2: Overview of ENTRUST allowing E2E security in publish/subscribe systems. Exemplary, the publisher joins for E2E security using a three step handshake.

Figure 2 shows an overview of ENTRUST, including an introduced trusted third-party, the key server, which manages the exchange of key material and is also connected to the broker as a PubSub client. The key server holds a pre-shared key (PSK) with all publishers and subscribers in the PubSub infrastructure and is tasked with sending specific *topic keys* to authorized clients which they can subsequently use to encrypt and decrypt payload transmitted via ordinary PubSub messages. While the PSK, mainly used for later authentication of the clients, is exchanged out-of-band before the deployment of the publishers and subscribers (Step ⓪, details in Section 5.1), the exchange of the topic keys requires an elliptic curve Diffie–Hellman (ECDH) to achieve forward secrecy before key transmissions (Step ① and ②, Section 5.2). After all publishers and subscribers received topic keys from the key server, the participants can communicate with full E2E protection (Section 5.3).

5.1 Bootstrapping: Out-of-Band Key Exchange

To enable clients, i.e., publishers and subscribers, and key server to authenticate each other for a subsequent exchange of topic keys, ENTRUST introduces a unique pairwise shared secret between each client and the key server. As shown in Step ⓪ in Figure 2, this procedure corresponds to an out-of-band communication where a pre-shared key (PSK) of various length is set on both, the key server and a new client, which wants to act as publisher or subscriber in the following. While this mechanism introduces a configuration overhead, it only needs to be performed once and can be executed either manually while setting up the client, e.g., configuring the broker's IP address, or via automated techniques such as NFC or QR codes, making the key server aware of a preinstalled PSK on the client [27, 34]. Thereby, ENTRUST does not require more complicated means, such as PKI infrastructures, to authenticate clients during operation increasing its efficiency.

As only key server and client possess a specific PSK, they can use this shared secret to authenticate each other in subsequent steps. To this end, both entities use the PSK during a subsequent handshake.

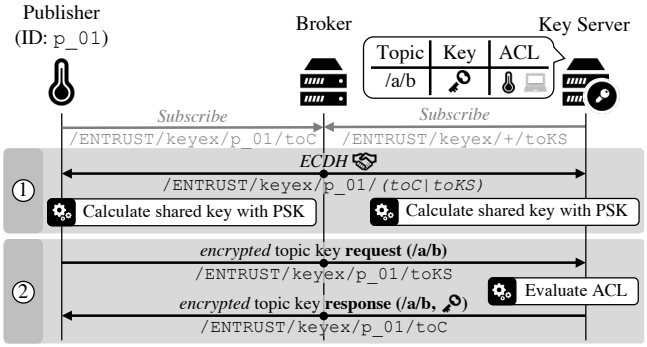

Figure 3: Subscription, handshake, and subsequent topic key transmission between a client (exemplary a publisher), and the key server. Here, + indicates a topic wildcard.

5.2 Facilitating a Topic Key Exchange

ENTRUST utilizes the key server to provide a symmetric key per topic. For message confidentiality (integrity), this *topic key* is used by publishers to encrypt (sign) and subscribers to decrypt (verify) messages on the corresponding topic. To this end, the key server securely sends the topic key via the PubSub protocol to requesting clients but ensures that only authorized clients can receive the key. To build a foundation to transmit topic keys to authorized clients securely, in ENTRUST every client performs an ECDH with the key server to generate fresh shared key material. Periodically refreshing this key material via subsequent ECDHs strengthens the information security ENTRUST introduces as once leaked key material does not compromise past or future sessions (forward secrecy). Subsequently, key server and client use this shared key material in combination with the PSK from the out-of-band key exchange to enable secure and authentic transmission of topic keys.

Key Exchange Topic: ENTRUST introduces a specific *key exchange topic* that is used for all communication between key server and clients (`/ENTRUST/keyex/`) to establish a secure topic key transmission on top of the existing, unmodified PubSub protocol. Each client has its own subtopic based on the client ID (`/ENTRUST/keyex/client_id/`), which the key server and this specific client use for individual authenticated and encrypted communication.

To ensure that the broker does not relay all sent messages back to the sender, ENTRUST separates all communication per direction in underlying subtopics, i.e., `/ENTRUST/keyex/client_id/toKS/` for messages sent from the client to the key server and `/ENTRUST/keyex/client_id/toC/` for communication from the key server to the client. The key server subscribes to all `toKS` subtopics, e.g., by using a wildcard for the client ID. Correspondingly, each client subscribes to the `toC` subtopic designated for its own client ID.

Handshake and Topic Key Negotiation: Figure 3 shows the handshake and topic key transmission after all entities subscribed to their designated key exchange topics. To perform topic key transmissions using fresh key material (to account for potentially compromised keys), key server and client perform a handshake (Step ①) before the key server provides topic keys (Step ②).

In general, the handshake bases on the well-established elliptic-curve Diffie–Hellman (ECDH) key exchange, as, e.g., used during a TLS PSK handshake [13]. To this end, a joining client, e.g., a publisher intending to publish data on a specific topic, starts to perform an ECDH key exchange with the key server via the dedicated key exchange topic for this client (e.g., `/ENTRUST/keyexchange/p_01/` for the client with client ID `p_01`). More precisely, the client sends an *ECDH key exchange request* to the `toKS` subtopic including its fresh ECDH public key. Then, the key server performs its half of the ECDH handshake and responds with an *ECDH key exchange response* including its calculated ECDH public key. Afterward, both entities are in possession of a ECDH shared secret to enable subsequent secure communication.

Mutual Authentication: To guarantee mutual authentication between the client and key server, ENTRUST requires both entities to derive a master secret from the ECDH shared secret in combination with the PSK exchanged in Step ⓪. Therefore, ENTRUST utilizes HKDF [28] as key derivation function to generate a 64 B master secret. To this end, HKDF is performed on the concatenation of the ECDH shared secret, its length, the PSK, and the PSK's length, similar as used in the well-established TLS PSK handshake [13]. Finally, key server and client use the first 32 B of the master secret to encrypt and authenticate messages from the key server to the client and the rest of the master secret to protect the communication in the opposite direction.

Topic Key Retrieval: Once the master secret has been established, the client can request the *topic key* from the key server for every topic it wants to subscribe or publish to in Step ②. To this end, the client publishes encrypted and authenticated *topic key requests*, containing the topic it requests the key for, in the designated subtopic for its communication with the key server. For encryption, ENTRUST relies on `ChaCha20` and for authentication on a `Poly1305` message authentication code (MAC) both known for good performance on constrained devices [8] that are prevalent in industrial scenarios. Subsequently, the key server checks whether the specific client is allowed to access the topics for which the client requested the topic keys using an access control list (ACL). To grant an authorized client access to a requested topic, the key server responds to the request with an encrypted and authenticated *topic key response* containing the topic and the corresponding topic key.

Secure Key Design: As potentially multiple publishers use the same symmetric topic key to encrypt messages within a specific topic and a reuse of an IV with the same key material weakens the protection inherently, ENTRUST has to ensure that the initialization vectors (IVs) used for encryption stay unique for all messages. To ensure this uniqueness, the *topic key response* includes a publisher-specific prefix (up to four byte) for all IVs used by this publisher.

Furthermore, from time to time, topic keys need to be renewed, e.g., when a publisher exhausts its available IVs or operators revoke the access rights of clients to specific topics. Therefore, clients can trigger the establishment of new topic keys by sending a *topic key renewal request* to the key server. When receiving such a request or when the key server decides to renew a topic key, e.g., due to revoked access rights or a timeout, the key server sends *topic key responses* containing new key material to all clients. To account for synchronization issues, e.g., subscribers receiving new key material before publishers, each key has an ID which publishers add to each published message and subscribers use to select the corresponding key material for decryption.

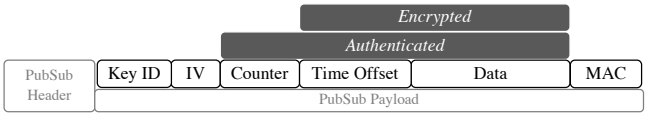

Figure 4: Data message structure proposed by ENTRUST. It allows E2E protected communication completely transparently for any underlying PubSub protocol.

5.3 Communication Protocol

As soon as subscribers and publishers possess the topic key for a specific topic, they can use this key to communicate securely benefiting from the E2E protection provided by ENTRUST transparently on top of the PubSub protocol. In this regard, ENTRUST uses the symmetric topic key to realize encryption and authentication for messages exchanged on the corresponding topic. Accounting for resource constraints prevalent in CPSs, especially in industrial settings, ENTRUST relies on a combination of the `ChaCha20` stream cipher for encryption and the `Poly1305` authenticator to generate a MAC [8] (as also used for topic key requests and responses in ENTRUST).

Message Structure: Figure 4 details how ENTRUST allows for transparent E2E-protected communication using an existing underlying PubSub protocol. Additionally introduced metadata, e.g., the key ID, an initialization vector, and a MAC, are inserted next to the sent encrypted data as payload within regular PubSub messages.

Message Reliability: Besides the need to provide E2E security (cf. R1 in Section 3.2), secure many-to-many communication in industrial CPS scenarios also requires that subscribers should be able to reliably detect duplication or dropping of messages (cf. R2 in Section 3.2). To enable subscribers to detect such duplication or dropping of messages, publishers in ENTRUST include a publisher-specific message counter into each message (reusing the unique prefix assigned to each publisher to prevent collisions of IVs). Using this counter, subscribers can reliably detect duplicated messages. Furthermore, they can reliably detect dropped messages as soon as the next message from the same publisher arrives. To additionally allow subscribers to detect dropped messages without relying on a subsequent message from the same publisher, publishers either announce at which time they will send the next message at latest or periodically publish (empty) heartbeat messages [23].

Takeaway: *With ENTRUST, the communication in PubSub systems is fully E2E-secured: from the publishers up to the subscribers. Thereby, it is transparently deployable without any changes to the underlying PubSub protocol admitting ENTRUST-enabled clients to piggyback and use already established PubSub infrastructure.*

6 SECURITY DISCUSSION

In this section, we discuss how ENTRUST realizes confidential, integrity-protected, and authenticated communication between all authorized clients in a PubSub network and thus meets the design requirements for E2E security in PubSub communication (especially R1 and R2, cf. Section 3.2).

To realize confidential communication, ENTRUST relies on per-topic keys that the key server distributes to all authorized clients (publishers and subscribers). To this end, the key server authenticates connecting clients using a client-specific pre-shared key (deployed on clients during their setup) and maintains an access control list specifying which client should have access to which topic. Publishers use the topic key received from the key server to symmetrically encrypt messages using `ChaCha20` and subscribers decrypt received messages accordingly. As only authenticated and authorized clients as well as the trusted key server possess the topic key, ENTRUST covers our requirements for confidential and authentic communication (R1). Importantly, the potentially untrusted broker does not have access to the topic key and thus cannot decrypt or modify exchanged messages. With ENTRUST we move this necessary trust to the separate key server which interacts securely with other entities as PubSub client only. Therefore, the key server must not be operated by potentially malicious cloud providers and does not require to offer any services directly to the Internet or production network reducing its attack surface.

As also forward secrecy is a desired property of E2E security schemes, ENTRUST does *not* use the pairwise pre-shared keys to encrypt communication between key server and clients. Instead, ENTRUST requires clients to perform an ECDH handshake with the key sever and from that point onward restrict the use of the pre-shared key to authentication.

To ensure authenticity and integrity for communication between clients, ENTRUST employs a `Poly1305` message authentication code (MAC). This MAC enables subscribers to reliably detect altered or injected messages and thus addresses the remaining aspect of our requirement for information security (R1).

Complementing the requirement of information security, i.e., confidentiality, integrity, and authenticity, E2E security for PubSub communication in industrial CPS scenarios also requires a reliable mechanism to prevent or detect the dropping and duplication of any messages (R2). To enable subscribers to detect duplicated or dropped messages reliably, ENTRUST introduces a publisher-specific message counter which is part of every message. After receiving a message, subscribers can use the message counter to detect duplicated messages (as the duplicated message would reuse an old counter value) as well as dropped messages (after the subsequent message from the same publisher has arrived). To also assist subscribers to detect dropped messages if no subsequent message from the same publisher arrives, publishers include a time offset into each message to signal at which point in time the next message from this publisher is to be expected. As CPSs typically communicate periodically, publishers simply can calculate the offset given their communication frequency but also can prevent exceeding timeouts of publishers by sending (empty) dummy messages.

Takeaway: *ENTRUST introduces secure E2E security for PubSub communication in CPSs by relying on well established cryptographic primitives and moving trust from the broker which originally is a sweetspot for attackers to a key server that only communicates as usual PubSub client. Thereby, ENTRUST covers all security requirements we identified for PubSub communication in CPSs.*

7 PERFORMANCE EVALUATION

The prevalent resource constraints in industrial CPS scenarios demand an efficient solution for E2E security in PubSub systems (R5). In the following, we show that ENTRUST is efficient enough to transparently operate on tightly resource-constrained CPS devices.

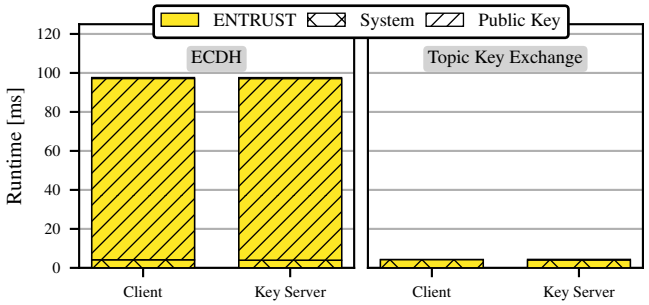

Figure 5: Runtime for ENTRUST ECDH and topic key exchange between client and key server. The regularly performed topic key exchange is very efficient but also the costs for the ECDH are well acceptable.

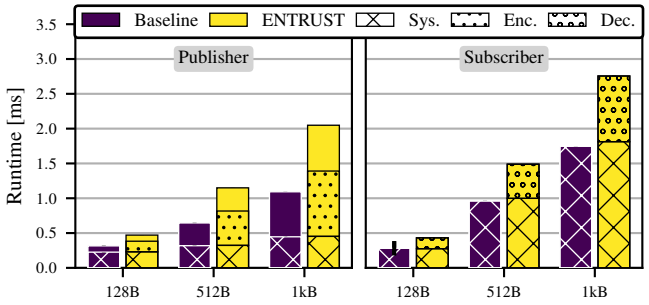

Figure 6: Comparison of ENTRUST's computation runtime with unprotected communication at publisher and subscriber split by different message sizes. Chosen lightweight cryptographic primitives allow for low runtime overhead.

Experimental Setup: For our evaluation, we implemented ENTRUST using the MQTT library for Photon [36] and the Arduino Cryptographic Library [52]. Furthermore, we use X25519 as elliptic curve for the ECDH and key sizes of 32 B as well an IV size of 8 B for ChaCha20. We deployed ENTRUST on three resource-constrained RedBear Duo (ARM Cortex-M3 @ 120 MHz) CPS devices, one acting as publisher, one acting as subscriber, and one acting as key server. Furthermore, we installed a `mosquitto` MQTT Broker [12] on a Raspberry Pi 3 (BCM2835 @ 1200 MHz). The Raspberry Pi is connected to a router via Ethernet and the Redbear Duos are connected via Wi-Fi 4 (802.11n).

With this testbed, we show the performance of ENTRUST for the different steps of our protocol design, i.e., initial handshake and topic key exchange (Section 7.1) as well as subsequent communication (Section 7.2). To this end, we measure the runtime of ENTRUST using the `micros()` Arduino function. We report on the arithmetic mean of 30 measurements and show 99 % confidence intervals. Finally, we study the impact of ENTRUST on the size of transmitted network packets (Section 7.3).

7.1 Handshake and Topic Key Exchange

We study the computation runtime required to perform a single handshake between client and key server as well as one topic key exchange. As shown in Figure 5 (left), the initial ECDH between the client and the key server introduces the same overhead of about 97 ms on both devices (client and key server). The measured overhead is dominated by the public key operations (93 ms) as well as the MQTT library and underlying system operations for communication (3.9 ms). As this initial ECDH needs to be performed only rarely, e.g., when a client starts up, this overhead of less than 100 ms to initially bootstrap the secure connection between client and key server is well acceptable even for large-scaled industrial CPS scenarios.

In comparison to the initial ECDH performed very rarely per client, the topic key exchange (TKE) has to be executed more often: (i) initially, once for every topic a client wants to use, and (ii) regularly, for each subscribed topic, depending on the frequency with which topic keys are renewed. Hence, by design, ENTRUST does not rely on any computationally expensive public key operations during the TKE and instead utilizes lightweight symmetric cryptographic primitives.

Consequently, as we illustrate in Figure 5 (right), the TKE is about two orders of magnitude more efficient than the ECDH: The runtime required for performing one TKE amounts to 4.3 ms at the client-side and 4.4 ms at the key server. For both entities, the runtime is dominated by system calls and the MQTT library. Notably, the cryptographic operations of ENTRUST only introduce a minor overhead, i.e., the encryption operation of the topic key request at the client (0.1 ms), the decryption at the key server (0.1 ms), and the corresponding response (key server: 0.1 ms, client: 0.1 ms) account for well less than 10 % of the total runtime.

All in all, our evaluation shows that both, the overhead induced by the initial handshake between clients and the key server as well as the TKE necessary to regularly exchange encryption keys for topics, can realistically be performed even by resource-constrained CPS devices in industrial scenarios.

7.2 Secure Communication

To realize E2E security for their PubSub-based communication, clients in ENTRUST use the received topic key to en- and decrypt messages using `ChaCha20` as well as `Poly1305` to authenticate and validate messages. In Figure 6, we study the resulting per-message processing overhead for publishing and subscribing to messages by comparing the computation runtime of ENTRUST to a baseline, corresponding to the computation runtime when using the same PubSub stack without any enabled security features. As we expect this runtime to depend on the size of transmitted messages, we perform the measurements for message sizes of 128 B, 512 B, and 1 kB to represent smaller and larger value clusters typically used in machine-to-machine communication [32].

Applying E2E security using ENTRUST when *publishing* messages (left half of Figure 6) introduces an overhead of 67 % (@ 128 B) to 53 % (@ 1 kB) when compared to the baseline, with a slightly smaller relative overhead for increasing message sizes. This overhead mainly stems from encryption and authentication of messages (0.2 ms @ 128 B), as the slight increase of messages sizes resulting from the security measures (cf. Section 7.3) does not noticeably increase the runtime of the MQTT library and system functionality. The runtime for other actions that are not affected by the application of ENTRUST, e.g., data generation, does not change.

Similarly, ENTRUST's overhead for processing messages at the *subscriber* (right half of Figure 6) is dominated by decryption and

verification operations (0.2 ms @ 128 B), while we do not observe an increase for the runtime of system functions and the MQTT library. Again, for increasing message sizes the relative overhead of ENTRUST also diminishes at the subscribers.

Overall, the high level of security achieved by transparently applying ENTRUST to sensitive communication of CPSs only introduces a modest increase in computation time which is well-manageable even by resource-constrained CPS devices.

7.3 Message Sizes

While generally designed to be lightweight and to only result in small communication overhead, ENTRUST still has to deal with the increased complexity of security functionality (in general). In particular, ENTRUST (i) introduces new message types and (ii) slightly increases the size of existing messages, e.g., by introducing a MAC and a per-publisher message counter, to realize its security functionality. In the following, we quantify and discuss the resulting impact on message sizes in more detail.

To bootstrap a secure connection between client and key server, ENTRUST requires an ECDH. Thereby, the exchange of the ECDH public keys requires two messages with a size of 34 B each.

For the subsequent and periodic topic key transmissions, ENTRUST introduces an encrypted and authenticated topic key request message, which has a minimum size of 28 B, amongst others consisting of an initialization vector and an authentication tag. Its eventual size depends on the length of the topic ID for which the client requests the key. The authenticated and encrypted topic key response has a minimum size of 60 B and also grows with the length of the topic ID the transmitted key corresponds to.

For the actual transmission of PubSub messages, ENTRUST adds a small overhead of 42 B to each message, including the initialization vector used when encrypting the message, an authentication tag, a per-publisher message counter, and the optional time offset until the next message is sent. While machine-to-machine messages often are JSON-encoded and therefore have a size of several bytes [32], the overhead added by ENTRUST in comparison is marginal.

Takeaway: *ENTRUST provides efficient E2E security for PubSub communication in CPSs at the cost of a few additional small packets for a lightweight handshake as well as a modest increase in total message size for regular PubSub messages. Especially for industrial settings with large JSON-encoded payloads that demand confidentiality and integrity for various reasons, these overheads are well-acceptable.*

8 CONCLUSION

The shift from isolated production and manufacturing deployments towards increasingly interconnected industrial networks—where many-to-many communication between CPSs now constitutes production's backbone—requires to fundamentally rethink communication security. So far, the broadly used PubSub communication scheme does not provide highly relevant security features for industrial deployments. Most importantly, there is no support for E2E security, exposing both security and safety of industrial deployments to the peril of misconfigured or malicious message brokers.

In this paper, we introduced ENTRUST, a scheme for k**e**y server-based e**n**d-**t**o-end secu**r**ity in p**u**blish/**s**ubscribe sys**t**ems which ensures confidentiality, integrity, and authenticity of exchanged messages as well as offers protection against dropping and duplication of messages. Thereby, it solely relies on the underlying PubSub protocol for communication and thus does not introduce any additional directly reachable services, which could increase the number of attack vectors in an industrial deployment.

ENTRUST does not require any changes to the message broker, thus promising easy deployability, and retains sender/receiver decoupling, which is an inherent part of PubSub communication to realize efficient many-to-many communication, especially in industrial CPS scenarios. To achieve all this and still account for prevalent resource constraints in industrial deployments, we solely rely on efficient cryptographic primitives such as ChaCha20-Poly1305.

With our work, we showed that it is indeed feasible to operate ENTRUST on resource-constrained CPS devices. The overhead for receiving key material from ENTRUST's trusted key server is marginal, while the extra effort required for encrypting and decrypting messages at publishers respectively subscribers is well invested considering the tremendous security benefits of ENTRUST.

For future work we envision to further improve the performance of ENTRUST, e.g., by using even more performant cryptographic primitives. For example, although ChaCha20 has shown to be very efficient on constrained CPS devices, AES might be more efficient when using hardware acceleration [40]. Furthermore, the authentication of messages can further be optimized, e.g., by utilizing progressive message authentication codes [1].

By proposing ENTRUST, we show that is possible to efficiently and effectively retrofit existing PubSub communication in industrial CPS deployments with end-to-end-security and thus allow modern CPSs to *securely* realize many-to-many communication.

ACKNOWLEDGMENTS

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy — EXC-2023 Internet of Production — 390621612.

REFERENCES

[1] Frederik Armknecht, Paul Walther, Gene Tsudik, Martin Beck, and Thorsten Strufe. 2020. ProMACs: Progressive and Resynchronizing MACs for Continuous Efficient Authentication of Message Streams. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. ACM, 211–223. https://doi.org/10.1145/3372297.3423349

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15, 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

[3] Andrew Banks and Rahul Gupta. 2014. MQTT Version 3.1.1. OASIS Standard.

[4] Smriti Bhatt, A Tawalbeh Lo'ai, Pankaj Chhetri, and Paras Bhatt. 2019. Authorizations in Cloud-Based Internet of Things: Current Trends and Use Cases. In *Proceedings of the 2019 4th International Conference on Fog and Mobile Edge Computing (FMEC '19)*. IEEE, 241–246. https://doi.org/10.1109/FMEC.2019.8795309

[5] Cristian Borcea, Yuriy Polyakov, Kurt Rohloff, and Gerard Ryan. 2017. PICADOR: End-to-end encrypted Publish–Subscribe information distribution with proxy re-encryption. *Future Generation Computer Systems* 71, 177–191. https://doi.org/10.1016/j.future.2016.10.013

[6] Markus Dahlmanns, Johannes Lohmöller, Ina Berenice Fink, Jan Pennekamp, Klaus Wehrle, and Martin Henze. 2020. Easing the Conscience with OPC UA: An Internet-Wide Study on Insecure Deployments. In *Proceedings of the ACM Internet Measurement Conference (IMC '20)*. ACM, 101–110. https://doi.org/10.1145/3419394.3423666

[7] Prajit Kumar Das, Sandeep Narayanan, Nitin Kumar Sharma, Anupam Joshi, Karuna Joshi, and Tim Finin. 2016. Context-Sensitive Policy Based Security in Internet of Things. In *Proceedings of the 2016 IEEE International Conference on Smart Computing (SMARTCOMP '16)*. IEEE. https://doi.org/10.1109/SMARTCOMP.2016.7501684

[8] Fabrizio De Santis, Andreas Schauer, and Georg Sigl. 2017. ChaCha20-Poly1305 Authenticated Encryption for High-Speed Embedded IoT Applications. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE '17)*. IEEE, 692–697. https://doi.org/10.23919/DATE.2017.7927078

[9] Tim Dierks and Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246. https://doi.org/10.17487/RFC5246

[10] Michael Dodson, Alastair R. Beresford, and Daniel R. Thomas. 2020. When will my PLC support Mirai? The security economics of large-scale attacks against Internet-connected ICS devices. In *Proceedings of the 2020 APWG Symposium on Electronic Crime Research (eCrime '20)*. IEEE.

[11] Dacfey Dzung, Martin Naedele, Thomas P. Von Hoff, and Mario Crevatin. 2005. Security for Industrial Communication Systems. *Proc. IEEE* 93, 6, 1152–1177. https://doi.org/10.1109/JPROC.2005.849714

[12] Eclipse Foundation. 2010. Eclipse Mosquitto. https://mosquitto.org/.

[13] Pasi Eronen and Hannes Tschofenig. 2005. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). IETF RFC 4279. https://doi.org/10.17487/RFC4279

[14] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. *Comput. Surveys* 35, 2, 114–131. https://doi.org/10.1145/857076.857078

[15] Jingcheng Gao, Jing Liu, Bharat Rajan, Rahul Nori et al. 2014. SCADA communication and security issues. *Security and Communication Networks* 7, 1, 175–194. https://doi.org/10.1002/sec.698

[16] Dennis J. Gaushell and Wayne R. Block. 1993. SCADA Communication Techniques and Standards . *IEEE Computer Applications in Power* 6, 3, 45–50. https://doi.org/10.1109/67.222741

[17] René Glebke, Martin Henze, Klaus Wehrle, Philipp Niemietz et al. 2019. A Case for Integrated Data Processing in Large-Scale Cyber-Physical Systems. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS '19)*. AIS, 7252–7261. https://doi.org/10.24251/HICSS.2019.871

[18] Lars Gleim, Jan Pennekamp, Martin Liebenberg, Melanie Buchsbaum et al. 2020. FactDAG: Formalizing Data Interoperability in an Internet of Production. *IEEE Internet of Things Journal* 7, 4, 3243–3253. https://doi.org/10.1109/JIOT.2020.2966402

[19] Robert Godfrey, David Ingham, and Rafael Schloming. 2014. OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. OASIS Standard.

[20] Maanak Gupta, Mahmoud Abdelsalam, Sajad Khorsandroo, and Sudip Mittal. 2020. Security and Privacy in Smart Farming: Challenges and Opportunities. *IEEE Access* 8, 34564–34584. https://doi.org/10.1109/ACCESS.2020.2975142

[21] Martin Henze. 2020. The Quest for Secure and Privacy-preserving Cloud-based Industrial Cooperation. In *Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS '20)*. IEEE. https://doi.org/10.1109/CNS48642.2020.9162199 Proceedings of the 6th International Workshop on Security and Privacy in the Cloud (SPC '20).

[22] Martin Henze, Roman Matzutt, Jens Hiller, Erik Mühmer et al. 2020. Complying with Data Handling Requirements in Cloud Storage Systems. *IEEE Transactions on Cloud Computing*. https://doi.org/10.1109/TCC.2020.3000336

[23] Martin Henze, Benedikt Wolters, Roman Matzutt, Torsten Zimmermann, and Klaus Wehrle. 2017. Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things. In *Proceedings of the 2017 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '17)*. IEEE, 185–192. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.236

[24] René Hummen, Hanno Wirtz, Jan Henrik Ziegeldorf, Jens Hiller, and Klaus Wehrle. 2013. Tailoring End-to-End IP Security Protocols to the Internet of Things. In *Proceedings of the 2013 21st IEEE International Conference on Network Protocols (ICNP '13)*. IEEE. https://doi.org/10.1109/ICNP.2013.6733571

[25] International Electrotechnical Commission. 2014. Industrial communication networks - Fieldbus specifications. IEC 61158.

[26] Yan Jia, Luyi Xing, Yuhang Mao, Dongfang Zhao et al. 2020. Burglars' IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP '20)*. IEEE, 465–481. https://doi.org/10.1109/SP40000.2020.00051

[27] Bernd Klauer, Jan Haase, Dominik Meyer, and Marcel Eckert. 2017. Wireless sensor/actuator device configuration by NFC with secure key exchange. In *2017 IEEE AFRICON*. IEEE, 473–478. https://doi.org/10.1109/AFRCON.2017.8095528

[28] Hugo Krawczyk and Pasi Eronen. 2010. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). IETF RFC 5869. https://doi.org/10.17487/RFC5869

[29] Sam Kumar, Yuncong Hu, Michael P. Andersen, Raluca Ada Popa, and David E. Culler. 2019. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. In *Proceedings of the 28th USENIX Security Symposium (SEC '19)*. USENIX Association, 1519–1536.

[30] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. 2014. Industry 4.0. *Business & Information Systems Engineering* 6, 4, 239–242. https://doi.org/10.1007/s12599-014-0334-4

[31] Pieter Maene, Johannes Götzfried, Ruan De Clercq, Tilo Müller, Felix Freiling, and Ingrid Verbauwhede. 2017. Hardware-Based Trusted Computing Architectures for Isolation and Attestation. *IEEE Trans. Comput.* 67, 3, 361–374. https://doi.org/10.1109/TC.2017.2647955

[32] Federico Maggi, Rainer Vosseler, and Davide Quarta. 2018. *The Fragility of Industrial IoT's Data Backbone: Security and Privacy Issues in MQTT and CoAP Protocols*. Technical Report. Trend Micro Research.

[33] Lukas Malina, Gautam Srivastava, Petr Dzurenda, Jan Hajny, and Radek Fujdiak. 2019. A Secure Publish/Subscribe Protocol for Internet of Things. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19)*. ACM. https://doi.org/10.1145/3339252.3340503

[34] Tobias Marktscheffel, Wolfram Gottschlich, Wolfgang Popp, Philemon Werli et al. 2016. QR code based mutual authentication protocol for Internet of Things. In *Proceedings of the 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM '16)*. IEEE. https://doi.org/10.1109/WoWMoM.2016.7523562

[35] Modbus-IDA. 2006. MODBUS Application Protocol Specification V1.1b.

[36] Hirotaka Niisato. 2014. MQTT for Photon, Spark Core. https://github.com/hirotakaster/MQTT.

[37] OPC Foundation. 2017. OPC Unified Architecture — Part 14: PubSub. OPC 10000-14: OPC Unified Architecture.

[38] Shrideep Pallickara and Geoffrey Fox. 2003. NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference (Middleware '03)*, Vol. 2672. Springer, 41–61. https://doi.org/10.1007/3-540-44892-6_3

[39] Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil, Geoffrey Fox, Yan Yan, and Yi Huang. 2006. A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems. In *Proceedings of the 2006 7th IEEE/ACM International Conference on Grid Computing (GRID '06)*. IEEE, 215–222. https://doi.org/10.1109/ICGRID.2006.311018

[40] Cristina Panait and Dan Dragomir. 2015. Measuring the performance and energy consumptionof AES in wireless sensor networks. In *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems (FedCSIS '15)*. IEEE, 1261–1266. https://doi.org/10.15439/2015F322

[41] Jan Pennekamp, Erik Buchholz, Yannik Lockner, Markus Dahlmanns et al. 2020. Privacy-Preserving Production Process Parameter Exchange. In *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC '20)*. ACM, 510–525. https://doi.org/10.1145/3427228.3427248

[42] Jan Pennekamp, René Glebke, Martin Henze, Tobias Meisen et al. 2019. Towards an Infrastructure Enabling the Internet of Production. In *Proceedings of the 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS '19)*. IEEE, 31–37. https://doi.org/10.1109/ICPHYS.2019.8780276

[43] Jan Pennekamp, Martin Henze, Simo Schmidt, Philipp Niemietz et al. 2019. Dataflow Challenges in an *Internet* of Production: A Security & Privacy Perspective. In *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy (CPS-SPC '19)*. ACM, 27–38. https://doi.org/10.1145/3338499.3357357

[44] Jan Pennekamp, Patrick Sapel, Ina Berenice Fink, Simon Wagner et al. 2020. Revisiting the Privacy Needs of Real-World Applicable Company Benchmarking. In *Proceedings of the 8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC '20)*.

[45] Eric Rescorla and Nagendra Modadugu. 2012. Datagram Transport Layer Security Version 1.2. IETF RFC 6347. https://doi.org/10.17487/RFC6347

[46] Rodrigo Roman, Jianying Zhou, and Javier Lopez. 2013. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks* 57, 10, 2266–2279. https://doi.org/10.1016/j.comnet.2012.12.018

[47] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted Execution Environment: What It is, and What It is Not. In *Proceedings of the 2015 IEEE IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '15)*, Vol. 1. IEEE, 57–64. https://doi.org/10.1109/Trustcom.2015.357

[48] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. 2015. Security and Privacy Challenges in Industrial Internet of Things. In *Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC '15)*. ACM. https://doi.org/10.1145/2744769.2747942

[49] Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. 2020. MQT-TZ: Secure MQTT Broker for Biomedical Signal Processing on the Edge. *Studies in Health Technology and Informatics* 270, 332–336. https://doi.org/10.3233/SHTI200177 Proceedings of 2020 Medical Informatics Europe on Digital Personalized Health and Medicine (MIE '20).

[50] Meena Singh, M. A. Rajan, V. L. Shivraj, and Purushothaman Balamuralidhar. 2015. Secure MQTT for Internet of Things (IoT). In *Proceedings of the 2015 5th International Conference on Communication Systems and Network Technologies (CSNT '15)*. IEEE, 746–751. https://doi.org/10.1109/CSNT.2015.16

[51] Anton V. Uzunov. 2016. A survey of security solutions for distributed publish/subscribe systems. *Computers & Security* 61, 94–129. https://doi.org/10.1016/j.cose.2016.04.008

[52] Rhys Weatherley. 2012. Arduino Cryptography Library. https://github.com/rweather/arduinolibs.

[53] Michael E. Whitman and Herbert J. Mattord. 2011. *Principles of Information Security* (4th ed.). Course Technology Press.