2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Application-level Security for ROS-based Applications

Bernhard Dieber[1], Severin Kacianka[2], Stefan Rass[3] and Peter Schartner[3]

Abstract— **While the topic of security in industrial applications has gained some momentum in recent years, there are still severe security vulnerabilities which are actively exploited for attacks. The robot operating system (ROS) is expected to further grow in usage and to be used in many industrial applications. Analysis, however, shows that it lacks several security enhancements in order to make it suitable for industrial use. In its current state, false data and commands can be injected posing a possible safety risk for the resulting product and humans in the production. In addition, data may be eavesdropped and used by outsiders to gain insight into the production process.**

In this paper we propose a security architecture intended for use on top of ROS on the application level. We use a dedicated authorization server to ensure that only valid nodes are part of the application. Cryptographic methods ensure data confidentiality and integrity. We show in a demonstration with a collaborative robot how our architecture can be used to secure a ROS-based application.

I. INTRODUCTION

Future industry production will be done in networked environments which require data being exchanged between the factory and data provisioning backend services. Since security can then no longer be provided by isolating networks from the outside, this will make the networked production sites accessible for cyber attacks. Production processes and the software controlling them are not sufficiently secure to withstand such attacks [1].

A well known example of an incident where an industrial control system was infected by malware is the Stuxnet worm [2]. Such attacks may be preceded by orchestrated malware infections designated to collect data [3], and/or exploit a lack of awareness on the user's side. While comprehensive standards exist to protect industrial control systems (such as, for example, being published by the NIST [4]), security in industrial control remains a demanding issue.

Speculating about reasons, an observable one may be a misconception of the threats related to industrial manufacturing plants. While it is often understood that blueprints or the manufacturing process must be kept confidential, other possible attack scenarios are easily overlooked: suppose that a hack on industrial robots causes them to place less than the minimal number of welding points to a car. If so, then the damage caused may range from shortened service intervals, up to lethal mass accidents. In such an attack, the target was not secret information, but causing maximal damage (to human life or at least the reputation of some enterprise).

ROS [5] has gained significant momentum in the past years. The publish/subscribe [6] pattern used by ROS is very useful to resolve the tight coupling and strong dependencies in a robotic system by providing various forms of decoupling and transparency. Exactly this transparency, however, also introduces several security risks which are not sufficiently addressed in ROS [7]. Publishers cannot control the consumption of their data and subscribers cannot easily verify the source and integrity of the received information. This weakness may be used to inject malicious information or for eavesdropping on sensitive data.

While using virtual private networks may seem to be an easy solution to secure the whole production network, this will also introduce additional overhead because all non-relevant data is encrypted as well. Furthermore it has already been proposed in [8] that a multi-layer approach to security should be considered in industrial networks. A VPN may be used in addition to our approach but securing the application itself should be a primary concern.

In this paper we introduce an application-level security architecture to overcome some major security threats which arise in a typical ROS application. We first describe our architecture (section II), then show its practical application (III), discuss it in section IV and present a variation for resource-constrained environments along with related work (section V).

II. APPLICATION-LEVEL SECURITY ARCHITECTURE

Throughout this work, we will confine ourselves to a non-invasive architecture, that is, we do not apply any changes to ROS, and implement security exclusively in the application layer. Thus, we treat ROS as a black-box in the following and implement security measures, such as an authentication server (AS) and dedicated functions in the ROS nodes themselves, on top of it. While this cannot cover all security risks (detailed in section IV), still the following crucial problems can be addressed nonetheless:

- a malicious publisher attempting to disturb the robot's trajectory using fake joint control commands
- a malicious subscriber to the robot state messages collecting the robots trajectory and actions and attempting to reverse-engineer the production process.

Whenever we speak about an "adversary" in the following, we mean a malicious publisher or subscriber (or an instance

*The work reported in this article has been supported by the Austrian Ministry for Transport, Innovation and Technology (bmvit) within the project framework Collaborative Robotics and by the Munich Center for Internet Research (MCIR).

[1]Bernhard Dieber is with the ROBOTICS institute at JOANNEUM RESEARCH bernhard.dieber@joanneum.at

[2]Severin Kacianka is with the Software Engineering Group at the Technical University of Munich kacianka@in.tum.de

[3]Stefan Rass and Peter Schartner are with the System Security Group in the Institute of Applied Informatics at the Alpen-Adria-University Klagenfurt {stefan.rass, peter.schartner}@aau.at

playing both roles); whichever is meant in particular will be clear from the context. Both types of hostile behavior can be counteracted by suitable authentication (of components and messages), as we describe in section II-A.

Referencing the security objectives presented in [9], our architecture covers confidentiality, integrity, authentication, authorization, auditability as well as third-party protection regarding the control command and state information exchanged in the ROS-based application. However, without a ROS-integrated solution we cannot provide availability and non-repudiation on the application level (see section IV for a detailed discussion).

Note, that the mechanisms described in the following sections are not only applicable to publishing and subscribing but can also be used to secure services in ROS.

A. Authentication and Key Agreement

In principle, the publish/subscribe system can be taken as a special form of broadcast encryption, where potentially many publishers of a certain message need to contact many (not explicitly known) subscribers in a secure and authentic way. While it is not too difficult to establish the necessary cryptographic operations, those nevertheless need to be incorporated at every step of the process. For this reason, we divide the architecture description into phases according to the lifecycle of a publisher and subscriber, and describe the relevant cryptographic operations per phase.

To start, let us assume that every possible message being transmitted by a publisher can be classified to fall into one out of a finite number N of topics. Let us reference these topics by indices $i \in \{1, 2, \ldots, N\}$ in the following. We will enforce a publisher to specify the topic (or several topics) from which messages are to be expected. This specification is done once during the registration, and then kept fixed for the lifetime of the publisher. Every other topic from a publisher will be rejected by the AS. The relevant details of the registration process are expanded in the next section.

Hereafter, we write $E(m, k)$ to mean the encryption (symmetric or asymmetric) of a message m under a key k. For asymmetric cryptography, we write pk, sk to mean the public and private key of an entity. For digital signatures, let $\texttt{sign}(m, sk)$ be the signature function taking a message m and private key sk to output a signature s. That signature s can be verified by a function $\texttt{verify}(s, pk) \in \{\texttt{true}, \texttt{false}\}$ that takes the public verification key as an additional input to the signature, and outputs either `true` or `false`, depending on whether or not the signature was cryptographically valid. We let our description in the following be abstract, yet emphasize that possible cryptographic schemes are AES for symmetric encryption, and RSA to handle asymmetric matters. The symbol $x \| y$ means the concatenation of the data items x and y in a way so that x and y can both be recovered uniquely from the compound representation $x \| y$. Usually, this will be a humble string concatenation, with a proper separator symbol.

Throughout this work, we assume that secret keys are protected from unauthorized access and securely stored within designated logical and physical parts of the infrastructure. Also, we will implicitly assume a secure boot process to reliably start a trusted version of the operating system. We will hereafter not explicitly go into details of such protection, and take it as subject to standard actions of key management.

1) Along with the registration request, P submits a public key certificate[a] $Z = (P, pk_P, S, \texttt{sign}(P \| pk_P \| S, sk_S))$ to the AS. In particular, the certificate Z thus tells the AS who the source S of the new publisher is.
2) The AS looks up the authentically stored public signature verification key of S, and verifies the certificate by checking if $\texttt{verify}(Z, pk_S) \stackrel{?}{=} \texttt{true}$. If and only if so, then it sends a random number r to P, which P digitally signs with its secret key sk_P that belongs to the public signature verification key pk_P. The new publisher candidate P then replies by sending the signature $sig = \texttt{sign}(r, sk_P)$ back to the AS.
3) As before, the AS takes the (authenticated) public key pk_P to verify that the signature on r is correct. That is, it accepts the new publisher P if and only if $\texttt{verify}(sig, pk_P) = \texttt{true}$.

[a]Note that the content of the certificate is intentionally restricted to only the relevant contents; the real certificates would have a much richer and complex structure.

Fig. 1. Simple Certificate Based Challenge-Response Authentication

B. Registration of a new Publisher

Any new publisher, upon registering itself with ROS, runs a challenge-response authentication to certify itself as a legitimate new publisher. This protocol is executed with the AS, whose duty is the validation and key management for the subsequent communication (publish/subscribe). The set of allowed (legitimate) publishers is assumed to be fixed and defined (hardwired) in the AS, as a set of certificates. To be specific, suppose each known (trusted) source S is known to the authentication server as a cryptographic (X.509) certificate, from which a public signature verification key pk_S can be extracted. Assume that a publisher P, affiliated to a trusted source S, wants to register itself in the AS, then it can only do so upon successful completion of the steps detailed in Figure 1. If P has authenticated itself to the AS, then the authentication is done in the opposite direction, to authenticate the AS to the new publisher. This avoids person-in-the-middle situations, where an intermediate malware could trick either the AS or the new publisher (or both) into sending their data and commands through a malicious proxy (which could then manipulate the information flow).

For direct, i.e., private point-to-point communication, it is easy to use the asymmetric keys known to both, the AS and P, to exchange a secret session key $K_{APP,P}$. Here, the AS acts as a trust-center (see [10, §13.2] for suitable protocols),

and handles the key establishment between the ROS-based application (APP) and the publisher. In general, all direct communication between APP and P must be encrypted under this key $K_{APP,P}$, and digitally signed using the respective sender's secret key (sk_{APP} for messages from APP to P, and sk_P for messages from P to APP).

C. Registration of a Subscriber

The registration of a subscriber proceeds analogous to that of a publisher, only with the obvious change of roles.

Additionally, the AS sends the subscriber a (digitally signed) list of public signature verification keys related to publishers of the message topics that the subscriber has registered for. The rationale is that every publisher is obliged to digitally sign its messages for authenticity, since a subscriber will drop messages in the following circumstances:

- the digital signature is missing or invalid
- the digital signature does not come from a previously known publisher. Note that looking up the signature verification key in the list given by the AS means that the AS has taken care of the identity check of the publisher previously. Thus, the subscriber's trust in the publisher is based on its trust in the AS (to have properly completed the authentication), and the trust in the digital signature.

D. Publishing

Assume that the new publisher P has registered to send messages under topic i (where i identifies some message topic). For each such message topic, the AS maintains an individual session key K_i. Every publisher that registers for messages of topic i is told the respective session key(s) K_i, under which it can encrypt its data and publish it to all subscribers. The subscriber, upon its registration for the same topic i, gets the same session key K_i from the AS.

It follows that a session key K_i becomes shared by possibly many publishers and subscribers. The process of changing these keys when a subscriber or publisher leaves is described later in section II-E.

If publisher P wants to broadcast a data item in an authenticated fashion, it completes the following tasks:

- it attaches its identity P to the data item m (belonging to topic i) and encrypts $P\|m$ under the session key K_i into a ciphertext $c = E(P\|m, K_i)$.
- it digitally signs the data item under its private signature key sk_P, thus getting a signature value $s = \texttt{sign}(c, sk_P)$.
- it attaches the topic i to the compound packet and broadcasts the tuple (i, c, s) to all subscribers.

Upon reception of a digitally signed message $M = (i, c, s)$, a subscriber parses M and completes the following steps:

1) it deciphers c using the known secret key K_i to retrieve the sender's identity P and the data item m (the correct session key i is indicated by the first entry in M). Note that this step is only possible if the subscriber has previously registered for messages of that particular topic i (the key K_i was told during the registration); if not, then M can be dropped immediately before any decryption attempts.
2) it verifies the digital signature s using the respective public key pk_P of the identity obtained in the first step (this spares the subscriber to work through the entire list of potential publishers for that message item).
3) it accepts the data item m if and only if m deciphered correctly under K_i, and the digital signature s on $E(P\|m, K_i)$ has been verified correctly.

E. Excluding Publishers or Subscribers

Note that under our restriction of not touching the operating system internals, we cannot easily exclude a publisher or subscriber, nor can we preclude a new publisher being started and replacing the existing one. However, the latter incident will either:

- introduce a new legitimate publisher (upon successful authentication), which may then correctly replace the current publisher,
- or end up with the publisher being registered to ROS, but not having received the proper keys from the AS, so that its messages will be abandoned by the subscribers.

In both cases, no immediate harm is to be expected, if a rejected publisher can be replaced by another legitimate one timely (in order to avoid a system failure or denial of service). Also, note that the AS can prevent a subscriber from effectively establishing itself in the system, since even despite its registration with ROS, it cannot read any of the encrypted contents if the AS terminated the registration process without sending the proper topic decryption keys.

Practical certificates being exchanged during the registration process, such as X.509, will of course have a much richer and more complex structure than sketched here, and in particular must contain the access and "publication rights" of the publisher. That is, the topics that a publisher may send messages for must be defined a priori in the list of permitted topics in the certificate that P submits to register itself. It may be allowed to register for fewer topics than the possible set, but a publisher cannot register for arbitrary topics. This is to avoid situations where a registered publisher becomes hijacked and starts sending forged messages (perhaps in disguise of other publishers).

If a publisher/subscriber is at any point discovered as malicious and shall be excluded from the system, then the AS needs to redistribute the topic keys across the remaining legitimate instances in the system. Standard broadcast encryption techniques apply here in a straightforward fashion (see e.g., [11]).

F. Additional Cryptographic Overhead

Since robot applications are expected to work in real time, it pays to measure the overhead induced by adding digital signatures and doing encryptions. We focus this assessment on the publishing and verification process, since the registration of a new publisher is part of a setup or startup phase and thus less time critical.

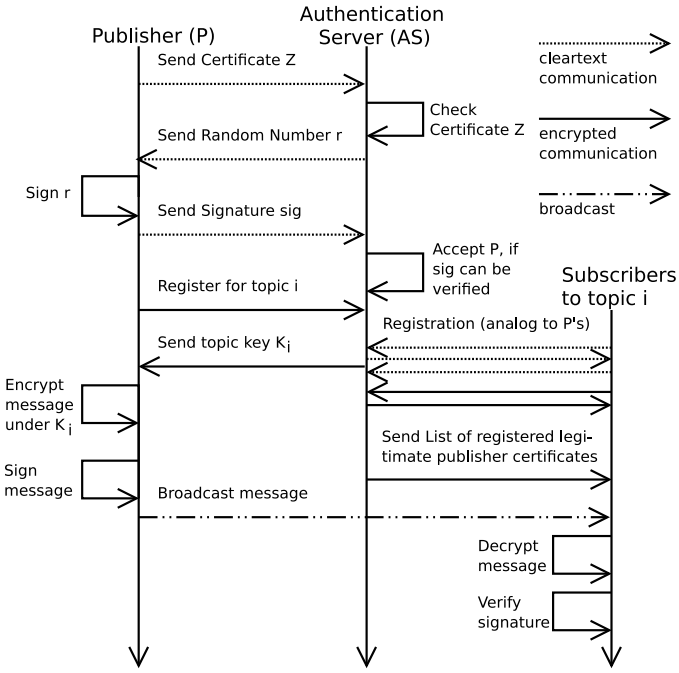

Fig. 2. System overview as sequence diagram

a) Additional data overhead: using proper block cipher modes (such as ciphertext stealing), the application of a block cipher E produces a ciphertext of exactly the same size as the input. Thus, the message m being published is essentially extended by the topic i, the publisher's identity P, as well as the digital signature s. Taking the bitlengths of i and P as negligibly small here, the major overhead is due to the signature. Using an elliptic curve digital signature algorithm with conteporary recommendations of key sizes (see [12]), the signature adds approximately 512 Bits of overhead per message. When RSA is used, the signature will be 2048 Bits.

b) Computational overhead: Reference [13] provides benchmarks of various cryptographic algorithms, including the ones proposed in this work. For the symmetric encryption E, say if we use AES, one encryption achieves a throughput of 109 MBit/s, i.e., the overhead is negligible for realistic lots of data. For RSA signatures, benchmarks using the Crypto++ Library measured 6.05ms for a signature creation (publisher), and 0.16ms for a signature verification (per subscriber).

G. Overview

Figure 2 shows an overview of the entire process, including the challenge-response authentication (from Figure 1), topic registration and message broadcasting. To complete the registration of a publisher, the AS is assumed to maintain a whitelist of trusted sources from which publishers (based on the information in the certificate presented for registration) are accepted (and rejected otherwise).

III. Demonstration

To demonstrate our approach we implemented a simple robot application in ROS. For our experiment we use ROS Indigo. Cryptographic functionality has been realized using

Fig. 3. The KUKA iiwa used in our experiments

CryptoPP[1]. In our application a KUKA iiwa (shown in Figure 3) is remotely controlled by a ROS node which periodically publishes joint angle positions. This will simulate a robot performing the same task over and over during its operation. In turn, the robot publishes its current state including its current joint angles as well as the cartesian position and rotation of the end-effector. Since the KUKA iiwa is a collaborative robot, humans may be nearby, who rely on the robot to cause no harm to them.

The joint commands published to the robot make it follow a simple trajectory to also visually indicate a correct behaviour of the application. A deviation from this trajectory will indicate that false movement commands have been injected by an adversary. In more complex cases where a truely collaborative robot is adapting its behaviour to a human or to a changed task, the visual observation of its path will no longer indicate a malfunction. Hence, automated methods to prevent and detect malicious behavior are needed.

For demonstration and evaluation, we compare the behavior of a robot without security measures (test case one) and with the security enhancements implemented as proposed in section II (test case two). For the first test case, a malicious publisher starts and tries to inject joint control commands which move the robot outside of the planned trajectory (possibly hitting workers, infrastructure or other robots).

In the second test case, we use a dedicated AS and have each node perform the procedure described in section II. Each joint value which is sent to the robot is encrypted using the topic encryption key. Thus, not the whole message but its individual components are encrypted. Figure III shows a model of all components and data types involved in this application. In addition, the iiwa Robot node encrypts the state information before publishing it. This prevents an unauthorized malicious subscriber from recording the robot's state data.

A. Application behavior with security disabled

With no additional security enabled, the malicious publisher node can interfere with the planned robot trajectory. It can send arbitrary movement commands which the robot will

[1]http://cryptopp.com/

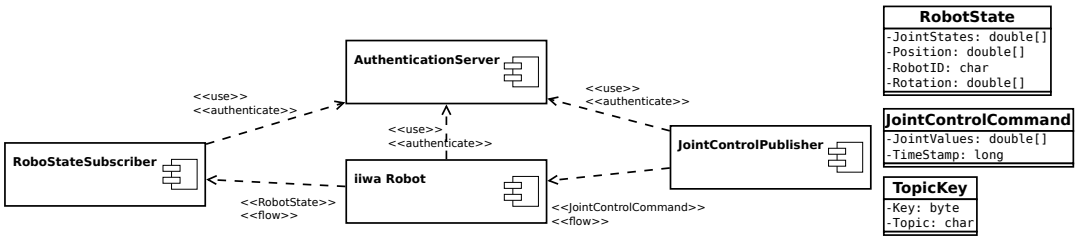

Fig. 4. A high-level view on the application architecture and the data types used.

execute. This includes unpredicable movements with high speed which may harm nearby humans or damage equipment and other robots.

B. Application behavior with security enabled

With enabled security, the malicious node cannot perform the authentication with the AS. Thus, it will not receive the topic key required to publish joint control commands. Unfortunately it cannot be prevented from publishing it despite having no key (this can only be prevented by ROS itself). Thus, the robot node will check incoming messages for valid encryption and will not execute any command which has not been encrypted. As a result, the iiwa will follow the authorized trajectory published by the trusted publisher and will ignore the joint commands published by the malicious node.

IV. Discussion

With the presented architecture we can avoid some of the most serious security vulnerabilities which are currently present in ROS. First, we can prevent unauthorized nodes from recording data which can be used for reverse engineering of production processes. This is achieved by topic-specific encryption keys which are only handed out to authorized application modules.

Second, we deal with the threat of unauthorized publishing to prevent the injection of false information into the robotic application. We accomplish this by verifying for each message that it has been encrypted using a valid key.

Still some insufficiencies persist which cannot be handled on the application level alone. They all need ROS itself to be modified. First, although the message content is encrypted and cannot be processed by unauthorized nodes, they can still collect information on which messages are published in which frequency. This could be solved by an end-to-end encryption of whole messages integrated into ROS itself. An alternative on the application level is to publish fake messages of certain types to disguise the true publishing frequency.

Second, in our approach we cannot prevent malicious publishers from publishing messages. We can only make sure that those messages are not interpreted by regular nodes. However, a denial-of-service attack with high publishing frequency could be possible.

Third, our approach cannot prevent a subscriber from subscribing to arbitrary topics. Thus, all messages of a certain topic will be delivered to it. Our approach only ensures that this subscriber cannot read the message contents without the proper decryption key.

A suitable method to prevent the exchange of keys between nodes must still be found. One approach is using code obfuscation to hide the key, and in addition, methods from leakage prevention (masking and hiding; see [14]) can be used to gain additional security against the theft of keys from code. A reliable protection of keys within unprotected software is doable by whitebox cryptography [15], [16], [17] however, these techniques are still in their infancies and have not reached a state of sufficient maturity to be used in our application by the time of this work.

V. Variants and Related Work

The security in industrial networks and applications has been an active topic for quite some time now [1], [4], [8], [9]. A general overview on the security issues in publish-subscribe systems has been presented in [18]. Recently, an intrusion detection method based on artificial intelligence methods for SCADA systems using special support vector machines [19]. Shin et al. [20] study various approaches for intrusion detection in wireless industrial networks and propose improvements. In [21] the authors present a method to extend classical wired industrial networks with wireless components while still preserving safety and security.

If a ROS-based application should be hardened where the involved entities have limited computational resources (such as may be expected for embedded systems), then costly operations like public key encryption or digital signatures should be avoided or must be at least minimized. A simple way to gain efficiency at the cost of some additional storage requirements for the publishers is the use of one-time passwords (see [10, Chp.10.2.5]), or the general preference of symmetric cryptography also for authentication purposes. One such scheme that extends this view even to distributed systems has been proposed in [22][2]. When occasional asymmetric cryptographic operations are permissible, then one-time password schemes can be constructed from hash-chains that use Chameleon hashes [23] as trapdoor one-way functions (we leave the details of such a construction for future work). Another interesting scheme that – like ours – uses challenge-response authentication to prove a robot's identity has been proposed in [24]. As with [22], cryptographic techniques must be used with care, and ad hoc "hand-crafted" solutions should in general be avoided.

[2]Alas, the MAC construction used in [22] has known weaknesses.

The work of [24] is nonetheless interesting in our context, as it clearly demonstrates the recognition of the authentication problem quite a while ago, whereas no solution on the level of the operating system seems to be available until today.

As a purely hardware based alternative to public key cryptography, one could implement a tamper-proof module for a subscriber that is dedicated to MAC verification only. If so, then the subscriber may verify MACs but cannot impersonate P as its hardware does not support MAC creation. In turn, this renders the computation of one-time password lists as described above unnecessary, and buys computational efficiency and reduced storage requirements at the cost of additional tamper-proof hardware (like a smartcard).

VI. Future Work

For cyber-physical systems a sound and complete a-posteriori account of the system's actions and decisions is necessary to improve the trust in such systems. During our present work we analyzed the default ROS log files and found that a reconstruction of event sequences leading up to a given situation is nearly impossible to do.

To alleviate this problem, we want to extend the logging mechanisms of ROS to allow us to infer *causal models* of messages and actions and thus allow us to reconstruct a system's behavior.

As a further next step we will transfer the concepts presented in this work to *roscore* to make it an integral part of ROS itself. This will help to deal with the problems which cannot be solved on the application level. A major part of this work will be to secure the communication between nodes and the master.

VII. Conclusion

Industrial applications are still insecure in certain settings, especially whenever the network is not isolated but connected to the outside world. ROS as one important software framework for future robot applications still has not sufficient security functionalities. In this paper we presented an approach for application-level security which eases the most severe security vulnerabilities in ROS-based applications. Using a dedicated authentication server we enable secure communication between ROS nodes.

With our approach we can prevent malicious nodes from injecting false commands into the process and we can ensure that data is only read by authorized parties. We have shown a demonstration of this concept in a practical project where a malicious node can be prevented from interfering with the original application.

References

[1] M. Cheminod, L. Durante, and A. Valenzano, "Review of security issues in industrial networks," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 277–293, Feb 2013.

[2] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," in *37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011)*, Nov 2011, pp. 4490–4494.

[3] N. Nelson, "The impact of dragonfly malware on industrial control systems," SANS Institute, Tech. Rep., 2016.

[4] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, "Guide to industrial control systems (ics) security," National Institute of Standards and Technology, Tech. Rep., 2015, NIST Special Publication 800-82, Revision 2.

[5] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, June 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[7] J. McClean, C. Stull, C. Farrar, and D. Mascareas, "A preliminary cyber-physical security assessment of the robot operating system (ros)," pp. 874 110–874 110–8, 2013. [Online]. Available: http://dx.doi.org/10.1117/12.2016189

[8] E. Byres, P. E. Dr, and D. Hoffman, "The myths and facts behind cyber security risks for industrial control systems," in *In Proc. of VDE Kongress*, 2004.

[9] D. Dzung, M. Naedele, T. von Hoff, and M. Crevatin, "Security for industrial communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1152–1177, June 2005.

[10] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[11] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," in *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 41–62.

[12] D. Giry, "Bluecrypt – cryptographic key length recommendation," http://www.keylength.com/, October 2016, accessed on: July 13th, 2016.

[13] W. Dai, "Crypto++ library 5.6," https://www.cryptopp.com/ benchmarks.html, 2016, accessed: 13th July, 2016.

[14] S. Mangard, M. E. Oswald, and T. Popp, *Power Analysis Attacks – Revealing the Secrets of Smart Cards*. Springer, 2007.

[15] gemalto SafeNet, "Understanding white box cryptography (white paper)," http://ru.safenet-inc.com/, 2016.

[16] M. Joye, "On white-box cryptography," *Security of Information and Networks*, 2008, trafford Publishing.

[17] J. Muir, "A tutorial on White-box AES," Cryptology ePrint Archive: Report 2013/104, 2013.

[18] C. Wang, A. Carzaniga, D. Evans, and A. Wolf, "Security issues and requirements for internet-scale publish-subscribe systems," in *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, Jan 2002, pp. 3940–3947.

[19] L. Maglaras and J. Jiang, "Intrusion detection in scada systems using machine learning techniques," in *Science and Information Conference (SAI), 2014*, Aug 2014, pp. 626–631.

[20] S. Shin, T. Kwon, G.-Y. Jo, Y. Park, and H. Rhy, "An experimental study of hierarchical intrusion detection for wireless industrial sensor networks," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 744–757, Nov 2010.

[21] J. Åkerberg, M. Gidlund, T. Lennvall, J. Neander, and M. Björkman, "Efficient integration of secure and safety critical industrial wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2011, no. 1, pp. 1–13, 2011. [Online]. Available: http://dx.doi.org/10.1186/1687-1499-2011-100

[22] R. Toris, C. Shue, and S. Chernova, "Message authentication codes for secure remote non-native client connections to ros enabled robots," in *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, April 2014, pp. 1–6.

[23] G. Ateniese and B. de Medeiros, "On the key exposure problem in chameleon hashes," in *Proceedings of the 4th international conference on Security in Communication Networks*, ser. SCN'04. Berlin, Heidelberg: Springer, 2005, pp. 165–179.

[24] W. Adi, "Mechatronic security and robot authentication," in *Bio-inspired Learning and Intelligent Systems for Security, 2009. BLISS '09. Symposium on*, Aug 2009, pp. 77–82.

Acronyms

APP ROS-based application
AS authentication server
ROS robot operating system