What is OPC-UA and how does it manage security?

Cisco IoT Security Research Lab
iot-security-labs@cisco.com
cisco.com/go/iotsecuritylab

Contents

Introduction

The market for industrial system components (programmable logic controllers, human-machine interfaces, supervision software, etc.) is characterized by a great diversity of vendors, protocols, and technologies. Initially, every component or application that wanted to communicate with a given product had to implement protocols specific to the product's vendor.

This situation had several drawbacks:

- the proliferation of proprietary protocols,
- high development costs in regard to the efforts made to interface with each product,
- the need for users to adopt specific vendor's technologies in order to avoid interoperability problems,
- the need to integrate upgrades for each proprietary protocol.

Additional difficulties included the need to manage different types of data within industrial communications: process data, alarms, events, history data, etc.

To overcome these difficulties, a group of five vendors (Fisher-Rosemount, Rockwell Software, Opto 22, Intellution, and Intuitive Technology) decided in 1994 to form the OPC Foundation[1] (Object Linking and Embedding for Process Control) to standardize communications within industrial systems.

The Foundation's goal was to define a solution that could ensure standardized data transfers between different types of components, particularly between monitoring applications (HMIs, SCADAs) in addition to field and control components (programmable logic controllers, RTUs, smart sensors and actuators, etc.)

At that time, monitoring applications mainly ran on Windows so the vendors opted for Microsoft OLE (Object Linking and Embedding) client/server technology, based on DCOM (Distributed Component Object Model).

DCOM is a technology that enables a client process running on a machine to enlist services implemented by a server process running on a remote machine. DCOM operates based on RPC (Remote Procedure Call) protocol to handle calls to remote procedures. The solution initially proposed by OPC was therefore heavily tied to Windows platforms from the outset.

In 1996, the OPC Foundation published its first specification called *OPC Data Access (OPC DA)*. With OPC DA, real-time reading and writing of data became possible without applications (HMIs, monitoring software, etc.) having any knowledge of the native protocol or data structures used by data sources (programmable logic controllers, smart sensors, etc.). Specifications for the management of alarms and events (*OPC A&E*), historical data (*OPC HDA*) or security (*OPC Security*) emerged later. All of these specifications are commonly referred to as *conventional OPC*.

OPC DA has a simple data model: each data access is characterized by a value, a quality and a timestamp. The value represents the data, the quality indicates its reliability, and the timestamp refers to its freshness. Data access is standardized using a client and server API.

Figure 1 depicts one example deployment of a *conventional OPC* solution within an industrial system. The OPC server is an application collecting data from control and field components, such as programmable logic controllers, via their native protocols (Modbus, IEC 61850, etc.). The server exposes that data using DCOM objects and methods, enabling multiple OPC clients (HMIs, SCADA) to indirectly read and write data in the control and field components through it.

[1] http://www.opcfoundation.org

An OPC DA client may either send requests to the OPC server to collect data or subscribe to the server to receive messages when specific data changes. Finally, besides data read and write services, the API also enables the discovery of OPC DA servers on the network and the listing of the information managed by each server.

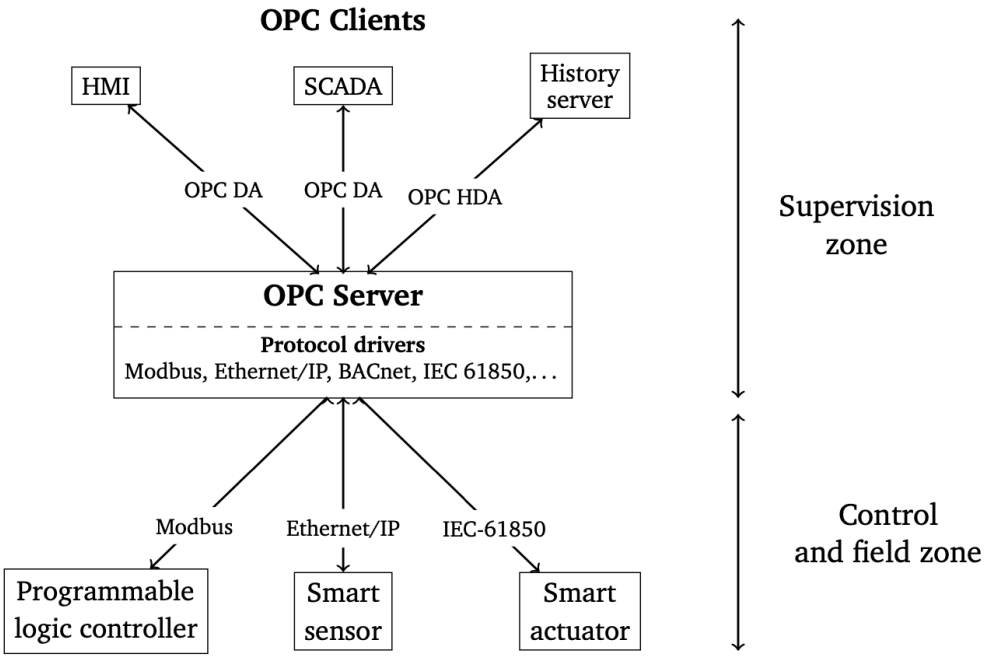

Figure 1. Example deployment of an OPC solution within an industrial system

However, *conventional OPC* suffers from several limitations. Since DCOM technology is almost only natively supported on Windows platforms, integrating an OPC server or client on an embedded machine (programmable logic controller, remote HMI) equipped with a real-time operating system has proven problematic. On Windows platforms, the complexity of DCOM configurations is a significant obstacle to the secure deployment of an OPC solution. In many cases, additional tunneling tools are needed to cross firewalls or bypass complex DCOM configurations.

When it comes to security, *conventional OPC* relies solely on the security offered by DCOM which is based on access control lists (ACLs). Three types of DCOM permissions can be configured:

- Launch permissions that define users or usergroups allowed to run a DCOM server on the remote server,
- Access permissions that control users or usergroups allowed to query a DCOM server while it is running,
- Configuration permissions that define users or usergroups allowed to edit the configuration of an OPC server in the registry.

The complex implementation of these ACLs, especially in cases where OPC clients and servers reside on remote machines and evolve in different security domains, may lead users to simply disable DCOM security measures by allowing, for example, remote access without authentication.

Another major problem with *conventional OPC* comes from the use of dynamic port allocation. *Conventional OPC* servers do not use a fixed TCP port because each process that exposes objects to clients is assigned a different TCP port. OPC clients must first connect to the server in order to see what TCP port is associated with a particular object. The clients then create a new TCP connection to the server. TCP ports that can be used by a

server can vary from port 1024 to 65535, which implies excessively permissive firewall configurations with a high security risk. Despite the recent development of firewalls that make it possible to overcome this difficulty by dynamically opening the necessary ports via OPC connection monitoring, their use requires additional investments from companies.

OPC-UA: Data model, services and architecture

In order to overcome the problems with *conventional OPC*, the OPC Foundation began defining a new specification called *OPC Unified Architecture* (*OPC-UA*), releasing the first version in 2006.

The main goal of *OPC-UA* is to specify a solution for communication between industrial deployments which can be easily deployed on different platforms. This new specification also aimed at preventing configuration and security problems related to *conventional OPC* while extending data representation via complex object-oriented modeling. *OPC-UA* combines all *conventional OPC* specifications (Data Access, Alarm & Events, Historical Data Access, etc.) into one specification. The most recent version of *OPC-UA* (v1.04) was released on November 22, 2017.

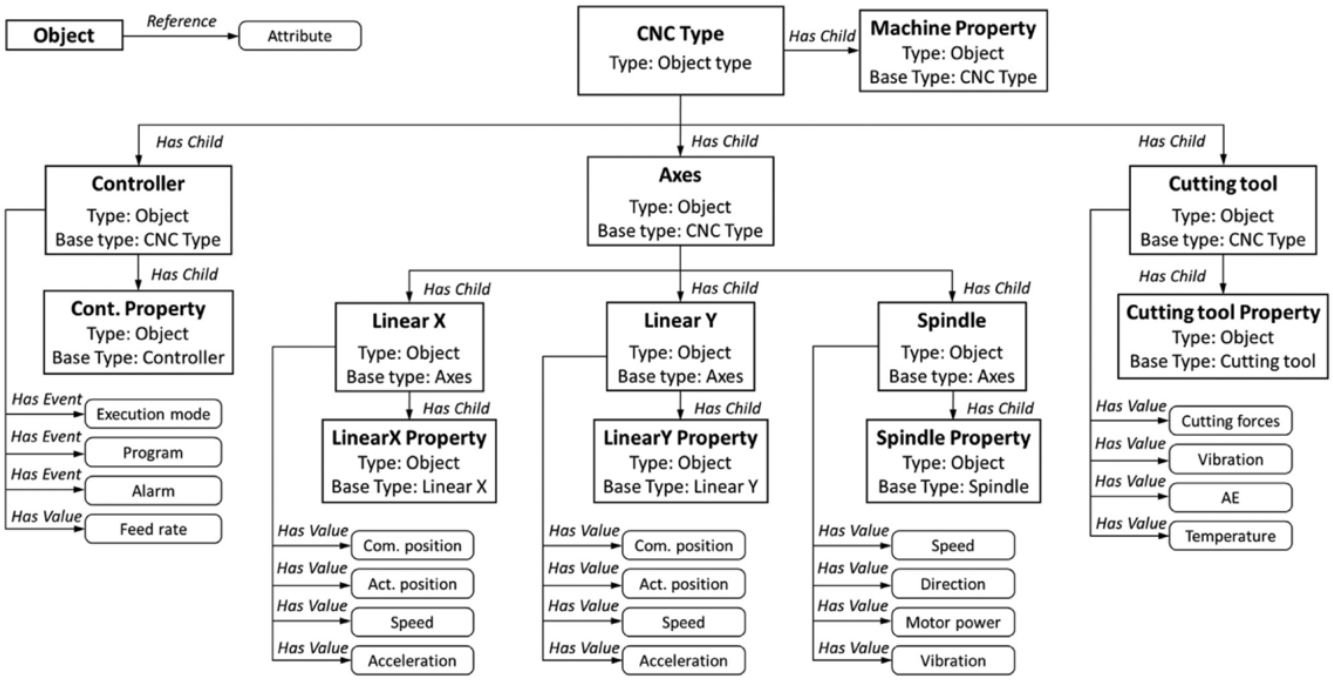

Figure 2. Example application of OPC-UA data model in a digitally controlled machining tower [1]

Data model

Information handled by an *OPC-UA* server is represented within a typed hierarchical model called *address space*. *Nodes* are the building blocks of the address space. Nodes are typed objects characterized by a set of attributes and one or more references to other nodes.

OPC-UA defines eight types of nodes, the main ones being the Object, Variable and Method types. Object nodes may represent a real physical system, a subsystem or a component. Variable nodes contain a value and may refer to actual data or properties of a node (metadata characterizing a node). Method nodes define the signature of methods that can be called from the *OPC-UA* interface. References between nodes within the address space

are also typed. Thus, a hasProperty reference binds a node to a property while a hasComponent reference represents a membership link between two nodes.

Finally, the notion of a *view* restricts the address space that can be accessed by clients based on their needs. Figure 2 provides an example application of the *OPC-UA* data model for a CNC lathe. The address space defines nodes that may correspond to actual entities (*Cuttingtool*, *Spindle*, etc.) or abstract ones (*LinearX Property*, *LinearY Property*, etc.). hasChild references make it possible to bind nodes in hierarchical relationships, while hasValue or hasEvent references are used to associate a node with an attribute (*Speed*, *Acceleration*, etc.).

Services

OPC-UA clients can invoke services implemented by *OPC-UA* servers. In total, *OPC-UA* defines 37 services, of which 21 are dedicated to establishing and managing communication infrastructure and 16 services relate to exchanging information. These services make it possible, among other things to:

- **Discover *OPC-UA* servers:** Using FindServers service, an *OPC-UA* client can obtain a list of accessible servers from a Discovery Server. To do so, *OPC-UA* servers must first be registered with the *Discovery Server*.
- **Read the security configuration of an *OPC-UA* server:** An *OPC-UA* server can listen to multiple Endpoints, with each Endpoint defining a listening URL, a transport protocol, a security mode, a security policy and its own certificate. GetEndpoints service can be used to get the list of Endpoints available on an *OPC-UA* server.
- **Establish a secure channel between an *OPC-UA* client and server:** Services such as OpenSecureChannel, ValidateCertificate, CreateSession, and ActivateSession make it possible to ensure the integrity and confidentiality of communications between an *OPC- UA* client and server via the creation of a secure channel and secure sessions. The algorithms used to encrypt and sign messages are specified by security policies exposed by *OPC-UA* server's Endpoints. The client then selects the Endpoint that matches the desired security policy. The procedure used to establish a secure communication channel is detailed later in the article.
- **Browse within the address space of an *OPC-UA* server:** Browse and BrowseNext services make it possible to explore nodes and references found in a *OPC-UA* server address space by specifying, for instance, a starting node and a selection filter.
- **Read and write attributes on an *OPC-UA* server:** Read and Write services make it possible to read nodes' attributes, including variables values.
- **Subscribe to data changes or events on an *OPC-UA* server:** As with *OPC DA*, clients can subscribe to the OPC server in order to get notifications when monitored data changes or when events occur. *OPC-UA* also makes it possible to control the quantity of data exchanged via deadband mechanisms (minimal variations of a variable are not propagated) or sampling (time interval to wait before propagating a variation).

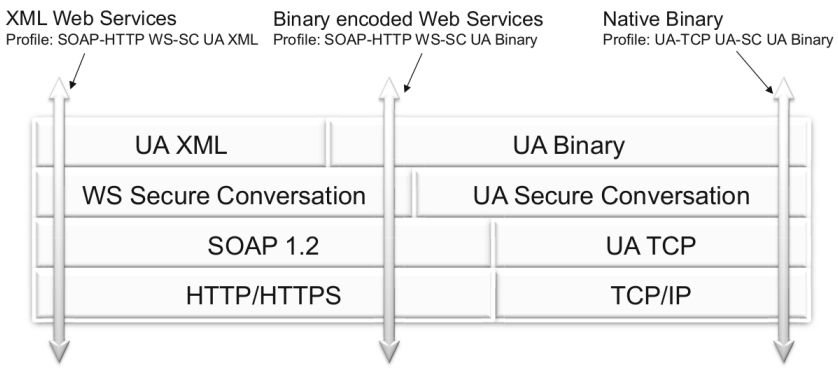

Figure 3. Application case for implementing OPC-UA services [2]

Since these services are abstractly specified, the *OPC-UA* standard introduces use cases that specify the data encoding and transport protocol to be used for services implementations (see Figure 3). Thus, in order to carry out simple communications via the Internet to enterprise management solutions such as ERPs (Enterprise Resource Planning), *OPC-UA* defines a use case based on SOAP (Simple Object Access Protocol) web services. Data can then be encoded in XML (simpler to implement) or in a binary format specific to *OPC-UA* (more compact and effective when transferring data, because it does not require an XML parser or an HTTP layer).

For cases where performance is critical, such as within the lower layers of an industrial system, the standard defines a technology use case based on *OPC-UA*'s native binary encoding with a transport protocol known as UA-TCP. This protocol, implemented on top of TCP, carries out mechanisms such as error management, making *OPC-UA* sessions more resilient to network interruptions. For instance, when an *OPC-UA* client loses a TCP connection, a new socket is instantiated and assigned to the preestablished secure channel, provided that it is re-authenticated by the server. This is the most common binary encoding that Cisco sees within its client networks.

OPC-UA: Security model

OPC-UA can be deployed at different levels within an industrial control system, with each of these levels potentially having different security and performance constraints. For instance, an OPC server deployed in the supervisory layer in order to collect data from the control layer will be less exposed to attacks than an Internet-accessible OPC server.

Likewise, because the lower layers of the industrial system are characterized by stricter real-time performance constraints than the supervisory or management layers, particular attention must be paid to the performance costs induced by the security measures. In order to adapt to these different settings, the security model adopted by *OPC-UA* is meant to be flexible and generic. This model is divided into two layers:

- **Communication layer** The role of this layer is to establish a secure channel between the client and the server. This channel provided the clients confidentiality, integrity and authentication through the use of encryption algorithms, signatures and digital certificates. When establishing a secure channel, the client and server agree on a security policy, i.e. common algorithms for the signature, encryption and key derivation.
- **Application layer** After establishing a secure channel, an OPC session is established to ensure user authentication and verify permissions according to a role system.

As with the services, *OPC-UA* specifies a generic security model which implementation varies based on the technological use cases (see Figure 3). Thus, for Web-services use cases (SOAP), *OPC-UA* is based on the *WS-SecureConversation* standard. On the other hand, for the use case based on binary encoding mapped onto the UA-TCP layer, *OPC-UA* defines a protocol inspired by *WS-SecureConversation* and *TLS* called *UA-SecureConversation*.

WS-SecureConversation

Developed by OASIS (Organization for the Advancement of Structured Information Standards)[2], *WS-SecureConversation* is a standard for making Web services-based data exchanges secure. *WS-SecureConversation* is used in conjunction with *WS-SecurityPolicy* to define security algorithms and *WS-Trust* for negotiating shared secrets when establishing a secure *OPC-UA* session. In order to encrypt and sign messages, standards such as *XML Encryption* and *XML Signature* are used. These standards are used to encrypt and sign all or part of an *XML document*.

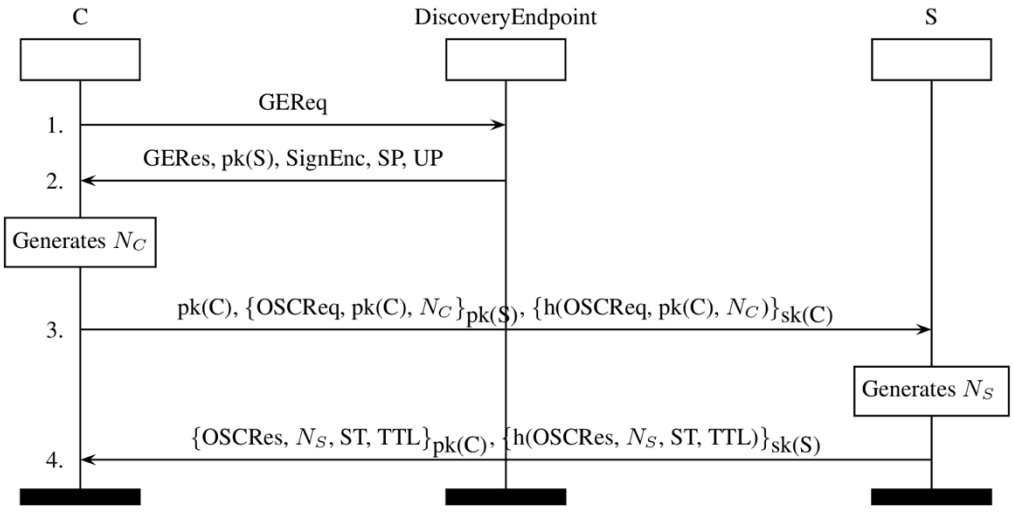

Figure 4. Sequence diagram showing the procedure for establishing a secure channel within the UA-SecureConversation protocol [3]

UA-SecureConversation

Establishing a secure connection via UA-SecureConversation takes four steps, as shown by the sequence diagram in Figure 4. In the first phase, if the *OPC-UA* client is not preconfigured with the information required to establish a secure connection, the client sends an unsecured GetEndpoints request (GEReq) to the Discovery Server in order to obtain security methods and protocols supported for each endpoint of the server, as well as their X.509 certificate, known as the Application Instance Certificate (pk(S)). The client then selects a security mode (None, Sign oe SignAndEncrypt) and checks the validity of the certificate provided by the Discovery Endpoint.

The use of X.509 certificates raises the issue of key management infrastructure within an *OPC-UA* facility. Although the use of self-signed certificates is an inexpensive alternative that is relatively well-suited to smaller industrial facilities, this solution is difficult to scale up, and may prove dangerous if the operator in charge is not experienced with managing X.509 certificates. The OPC Foundation recommends choosing a reliable certification authority (CA), and using a Global Discovery Server (GDS) to automatically manage certificates [4].

[2] https://www.oasis-open.org

A GDS is an *OPC-UA* server that has, among other things, a trusted list of certification authorities and their certificates.

In the second phase, depending on the security mode selected, the client sends a query in order to open a secure communication channel. If the chosen security mode is None, the rest of the procedure for establishing a secure channel is not applicable, as this mode is primarily there for compatibility reasons. If the chosen mode is SignAndEncrypt, the query message sent contains three parts [3]:

- The client's Application Instance Certificate, including its public key pk(C)
- A OpenSecureChannelRequest (OSCReq) message, m_c, containing a nonce N_C generated by the client, as well as the client's certificate. This message is encrypted using the public key of the server pk(S)
- The hash of the message m_c signed by the private key of the client sk(C)

The server then checks the validity of the client certificate, then decrypts and verifies the query message's signature. The server then sends a response message containing:

- A OpenSecureChannelResponse (OSCRes) message, m_s, containing (i) a nonce N_S generated by the server, (ii) a security token that represents a single identifier for that session, and (iii) a TTL (TimeToLive) piece of information specifying the message's validity period. This message is encrypted using the public key of the client pk(C)
- The hash of m_s signed by the private key of the server sk(S)

At the end of this exchange, the client and server can derive the symmetrical encryption keys by applying a hash function to the two nonces N_C and N_S. These keys are used both to encrypt the messages within the established communication channel, as well as to guarantee the integrity of the messages using a message authentication code (MAC).

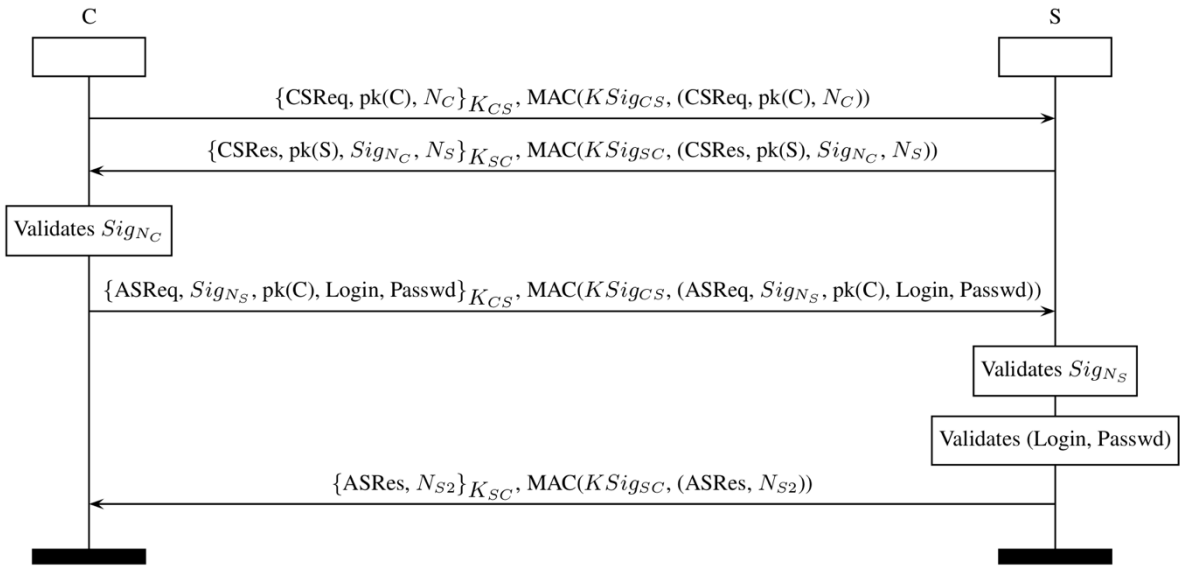

Figure 5. Sequence diagram showing the procedure for establishing a secure session within the UA-SecureConversation protocol [3]

After establishing a secure communication channel, the client sends a CreateSessionRequest query in order to establish a session on the secure channel (Figure 5). That query contains:

- A CreateSessionRequest (CSReq) message, m'_c, containing a N_C nonce generated by the client, as well as its public key. This message is encrypted using K_{CS}, one of the symmetrical keys derived when establishing the secure channel.
- The MAC of the message m'_c calculated using a $KSig_{CS}$ key derived when establishing the secure channel

The nonce NC enables the client to ensure that the server is the same one that established the secure communication. As a response, the server sends:

- A CreateSessionResponse (CSReq) message, m'_s, containing the nonce N_C signed using the private key of the server sk(S) ($Sig_{N_C} = \{pk(C), N_C\}_{sk(S)}$), as well as a nonce N_S generated by the server. This message is encrypted using K_{SC}, one of the symmetrical keys derived when establishing the secure channel. Here, Sig_{N_C} is the server's response to the client's challenge.
- The MAC of the message m'_s calculated using a $KSig_{SC}$ key derived when establishing the secure channel

The nonce N_S enables the server to ensure that the client is the same entity that had previously established the secure channel. To do so, the client sends:

- A ActivateSessionRequest (ASReq) message, m'', containing the nonce N signed using $_cS$ the private key of the client sk(C) ($Sig_{N_S} = \{pk(S), N_S\}_{sk(C)}$) as well the client's Login and Password. This message is encrypted using K_{CS}. Here, Sig_{N_S} is the client's response to the server's challenge.
- The MAC of the message m'' using $KSig_{c\,CS}$

After validating the signature of the nonce N_S as well as the client's identifiers, the server sends a ActivateSessionResponse (ASRes) message containing a new nonce N_{S2} that the client can use in order to reset the session in case of a timeout. At the end of these exchanges, a secure session is established between the client and the server.

Conclusion

The enhanced data model offered by OPC-UA makes it potentially suitable for complex areas such as power grids [2] or connected industrial objects [5]. OPC-UA has therefore been adopted by groups working to standardize communications, such as MDIS (Master Control Station [MCS] - Distributed Control System (DCS) Interface Standardization) that bring together gas and oil companies, as well as Open-SCS from pharmaceuticals.

However, questions persist concerning the security of OPC-UA. Analyses conducted on the OPC-UA specification and on reference implementations show the presence of several weaknesses [6, 7]. Being a protocol specification, OPC-UA does not provide constraints on pseudorandom number generators, and does not explicitly prohibit the use of certain cryptographic algorithms that are considered weak, such as SHA1. The secure channel establishment protocol does not ensure that communications will remain confidential.

As a result, the discovery by an adversary of the nonces exchanged when establishing a secure channel compromise the confidentiality of messages exchanged via that channel. Likewise, formal analyses conducted on the secure channel establishment protocol UA-SecureConversation show that man-in-the- middle attacks are possible in some security modes [3]. Finally, analyses of certain reference implementations prove the insufficiency of verifications in the sequence numbers used within the sessions, thereby opening the door to session hijacking or message replay attacks [7].

References

[1] Chao LIU et al. “A systematic development method for cyber-physical machine tools”. In: Journal of Manufacturing Systems (Feb. 2018).

[2] S. LEHNHOFF et al. “OPC Unified Architecture: A Service-oriented Architecture for Smart Grids”. In: 2012 First International Workshop on Software Engineering Challenges for the Smart Grid (SE-SmartGrids). June 2012, p. 1-7.

[3] Maxime PUYS, Marie-Laure POTET and Pascal LAFOURCADE. “Formal Analysis of Security Properties on the OPC-UA SCADA Protocol”. In: Computer Safety, Reliability, and Security. Under dir. of Amund SKAVHAUG, Jérémie GUIOCHET and Friedemann BITSCH. Cham: Springer International Publishing, 2016, p. 67-75.

[4] OPC FOUNDATION. Practical Security Recommendations for Building OPC UA applications. https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Security- Advise-EN.pdf

[5] OPC FOUNDATION. Microsoft Contributing Open-Source OPC UA. https://opcconnect. opcfoundation.org/2017/06/microsoft-contributing-open-source-opc-ua/

[6] KASPERSKY. OPC-UA Security Analysis. https://ics-cert.kaspersky.com/reports/2018/ 05/10/opc-ua-security-analysis/

[7] FEDERAL OFFICE FOR INFORMATION SECURITY (BSI). OPC-UA Security Analysis. https://opcfoundation.org/wp-content/uploads/2017/04/OPC_UA_security_analysis-OPC-F-Responses-2017_04_21.pdf

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **https://www.cisco.com/go/offices**.

April 2021