# Formal Analysis of Security Properties on the OPC-UA SCADA Protocol

Maxime Puys[1], Marie-Laure Potet[1], and Pascal Lafourcade[1,2]

(1) Verimag, University Grenoble Alpes, Saint-Martin-d'Hères, France
firstname.lastname@imag.fr
(2) LIMOS, University Clermont Auvergne, Campus des Cézeaux, Aubière, France
pascal.lafourcade@udamail.fr

**Abstract.** Industrial systems are publicly the target of cyberattacks since Stuxnet [1]. Nowadays they are increasingly communicating over insecure media such as Internet. Due to their interaction with the real world, it is crucial to prove the security of their protocols. In this paper, we formally study the security of one of the most used industrial protocols: OPC-UA. Using ProVerif, a well known cryptographic protocol verification tool, we are able to check secrecy and authentication properties. We find several attacks on the protocols and provide countermeasures.

## 1 Introduction

Industrial systems also called SCADA (*Supervisory Control And Data Acquisition*) have been known to be targeted by cyberattacks since the famous Stuxnet case [1] in 2010. Due to the criticality of their interaction with the real world, these systems can potentially be really harmful for humans and environment. The frequency of such attacks is increasing to become one of the priorities for governmental agencies, *e.g.* [2] from the US National Institute of Standards and Technology (NIST) or [3] from the French *Agence Nationale de la Sécurité des Systèmes d'Information* (ANSSI).

Industrial systems differ from other systems because of the long lifetime of the devices and their difficulty to be patched in case of vulnerabilities. Such specificities encourage to carefully check standards and applications before deploying them. As it already appeared for business IT's protocols for twenty years, automated verification is crucial in order to discover flaws in the specifications of protocols before assessing implementations. However, the lack of formal verification of industrial protocols has been emphasized in 2006 by Igure *et al.* [4] and in 2009 by Patel et al. [5]. They particularly argued that automated protocol verification help to understand most of the vulnerabilities of a protocol before changing its standards in order to minimize the number of revisions which costs time and money.

**State-of-the-art.** Most of the works on the security of industrial protocols only rely on specifications written in human language rather than using formal methods. In 2004, Clarke *et al.* [6] discussed the security of DNP3 (*Distributed Network Protocol*) and ICCP (*Inter-Control Center Communications Protocol*). In 2005, Dzung *et al.* [7] proposed a detailed survey on the security in SCADA systems including informal analysis

on the security properties offered by various industrial protocols: OPC (*Open Platform Communications*), MMS (*Manufacturing Message Specification*), IEC 61850, ICCP and EtherNet/IP. In 2006, in the technical documentation of OPC-UA (*OPC Unified Architecture*) the authors detailed the security measures of the protocol (specially in part 2, 4 and 6). In 2015, Wanying *et al.* [8] summarized the security offered by MODBUS, DNP3 and OPC-UA.

On the other hand, some works propose new versions of existing protocols to make them secure against malicious adversaries. In 2007, Patel *et al.* [9] studied the security of DNP3 and proposed two ways of enhancing it through digital signatures and challenge-response models. In 2009, Fovino *et al.* [10] proposed a secure version of MODBUS relying on well-known cryptographic primitives such as RSA and SHA2. In 2013, Hayes *et al.* [11] designed another secure MODBUS protocol using hash-based message authentication codes and built on STCP (*Stream Transmission Control Protocol*). To the best of our knowledge, Graham *et al.* [12] is the only work directly using formal methods to prove the security of industrial protocols or find attack against them. They proposed a formal verification of DNP3 using OFMC [13] (Open-Source Fixed-Point Model-Checker) and SPEAR II [14] (Security Protocol Engineering and Analysis Resource).

**Contributions.** We propose a formal analysis of the security of the sub-protocols involved in the OPC-UA handshake, namely OPC-UA *OpenSecureChannel* and OPC-UA *CreateSession*. These sub-protocols are crucial for the security since the first aims at authenticating a client and a server and deriving secret keys while the second allows the client to send his credentials to the server. To perform our security analysis, we use one of the most efficient tools in the domain of cryptographic protocol verification according to [15], namely ProVerif developed by Blanchet *et al.* [16]. It considers the classical Dolev-Yao intruder model [17] who controls the network, listens, stops, forges, replays or modifies some messages according to its knowledge. The perfect encryption hypothesis is assumed, meaning that it is not possible to decrypt a ciphertext without its encryption key or to forge a signature without knowing the secret key. ProVerif can verify security properties of a protocol such as secrecy and authentication. The first property ensures that a secret message cannot be discovered by an unauthorized agent (including the intruder). The authentication property means that one participant of the protocol is guaranteed to communicate with another one. Modeling credential in ProVerif is not common and requires to understand the assumptions made in the protocol in order to model it correctly. We follow the official OPC-UA standards in our models and checked it against a free implementation called *FreeOpcUa*[1]. Finally, using ProVerif, we automatically find attacks against both sub-protocols and provide simple realistic countermeasures. All sources we developed are available[2].

**Outline.** In Section 2, we analyze the security of OPC-UA *OpenSecureChannel* and OPC-UA *CreateSession* in Section 3. Finally, we conclude in Section 4.

[1] https://freeopcua.github.io/

[2] http://indusprotoverif.forge.imag.fr/PPL16.tar.gz

## 2 OPC-UA OpenSecureChannel

The *OpenSecureChannel* sub-protocol aims to authenticate a client and a server and allows them to exchange two secret nonces (random numbers) that will be used to derive shared keys for the later communications. Moreover, OPC-UA can be used with three security modes, namely *None*, *Sign* and *SignAndEncrypt*.

- *SignAndEncrypt*: messages are signed $\{h(m)\}_{sk(X)}$ and encrypted $\{m\}_{pk(X)}$, where $h$ is an hash function, $sk(X)$ the secret key associated to $X$ and $pk(X)$ the public key of $X$. This mode claims to provide secrecy of communication using symmetric and asymmetric encryption, but also both authentication and integrity through digital signatures.
- *Sign*: it is the same as SignAndEncrypt but messages are only signed $\{h(m)\}_{sk(x)}$, and not encrypted.
- *None*: using this mode, the *OpenSecureChannel* sub-protocol does not serve much purpose as it does not provide any security but is used for compatibility.

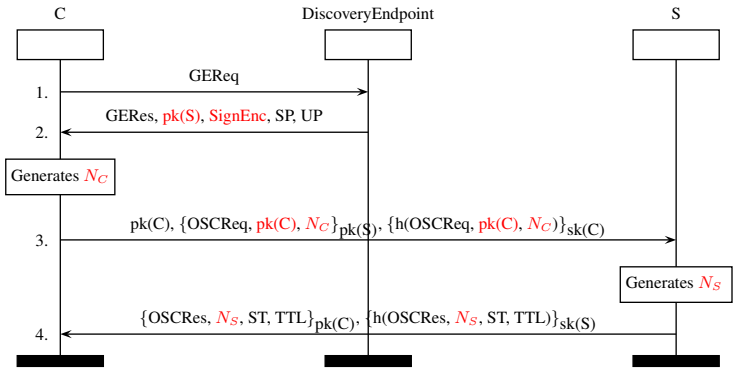


**Fig. 1.** OPC-UA *OpenSecureChannel* sub-protocol in mode SignAndEncrypt.

This protocol is described in Figure 1. In message 1, $C$ requests information on $S$ with *GEReq* meaning *GetEndpointRequest*. In message 2, $DiscoveryEndpoint$ answers with the following information where *GERes* stands for *GetEndpointResponse*, *SP* for *Security Policy* and *UP* for *UserPolicy*. Both *SP* and *UP* are used for cryptographic primitive negotiations. In message 3, $C$ sends a nonce $N_C$ to $S$ with *OSReq* standing for *OpenSecureChannelRequest*. Finally in message 4, $S$ answers a nonce $N_S$ to $C$ with *OSCRes* for *OpenSecureChannelResponse*, *ST* for *SecurityToken* (a unique identifier for the channel) and *TTL* for *TimeToLive* (its life-time). The four terms *GEReq, GERes, OSCReq* and *OSCRes* indicate the purpose of each message of the protocol. At the end of this protocol, both $C$ and $S$ derive four keys ($K_{CS}$, $KSig_{CS}$, $K_{SC}$ and $KSig_{SC}$) by hashing the nonces with a function named $P\_hash$, similar as in TLS [18]: $(K_{CS}, KSig_{CS}) = P\_hash(N_C, N_S)$ and $(K_{SC}, KSig_{SC}) = P\_hash(N_S, N_C)$.

### 2.1 Modeling

Normally, a *GetEnpointRequest* would be answered by a list of session endpoints with possibly different security modes. We suppose that the client always accepts the security mode proposed. Client's and server's certificates are modeled by their public keys. Moreover, thanks to the perfect encryption hypothesis, we can abstract the cryptographic primitives used. We consider an intruder whose public key would be accepted by a legitimate client or server. Such an intruder could for instance represent a legitimate device that has been corrupted through a virus or that is controlled by a malicious operator. We consider the following security objectives: (i) the secrecy of the keys obtained by $C$ (denominated by $K_{CS}$ and $KSig_{CS}$), (ii) the secrecy of the keys obtained by $S$ (denominated by $K_{SC}$ and $KSig_{SC}$), (iii) the authentication of $C$ on $N_C$ and (iv) the authentication of $S$ on $N_S$.

### 2.2 Results

We model in ProVerif this protocol for the three security modes of OPC-UA for each objective proposed. Results provided by ProVerif are shown in Table 1.

| OPC-UA Security mode | Objectives | | | |
|---|---|---|---|---|
| | Sec $K_{CS}$ | Sec $K_{SC}$ | Auth $N_S$ | Auth $N_C$ |
| None | UNSAFE | UNSAFE | UNSAFE | UNSAFE |
| Sign | UNSAFE | UNSAFE | UNSAFE | UNSAFE |
| SignEnc | SAFE | SAFE | UNSAFE | UNSAFE |

**Table 1.** Results for *OpenSecureChannel* sub-protocol

Obviously, as the security mode None does not provide any security, all objectives can be attacked. Moreover, as nonces are exchanged in plaintext in security mode Sign, the keys are leaked. Finally, in the case of Sign and SignAndEncrypt, the intruder reroutes messages to mount attacks on authentication in order to bypass replay protections such as timestamps as the packet's destination is changed rather than being replayed later. Figure 2 shows an attack on the authentication of $C$ using $N_C$. This attack is possible because the standard OPC-UA protocol does not require explicitly to give the identity of the receiver of a message. Thus it allows the intruder to send to $S$ the signed message $C$ sent to him similarly as the *man-in-the-middle* attack on the Needham-Schroeder protocol [19].

### 2.3 Fixed version

We propose a fixed version of the *OpenSecureChannel* sub-protocol using one of the classical counter-measures for communication protocols proposed in [20]. It consists in explicitly adding the public key of the receiver to the messages and thus avoiding an intruder to reroute signed messages to usurp hosts, as presented in Section 2.2. This resolves the authentication problem but, as ProVerif confirms, attacks on secrecy are still present. In order to solve the remaining secrecy attacks, we use the key wrapping [21] mechanism present in the OPC-UA standards [22–25]. All occurrences of

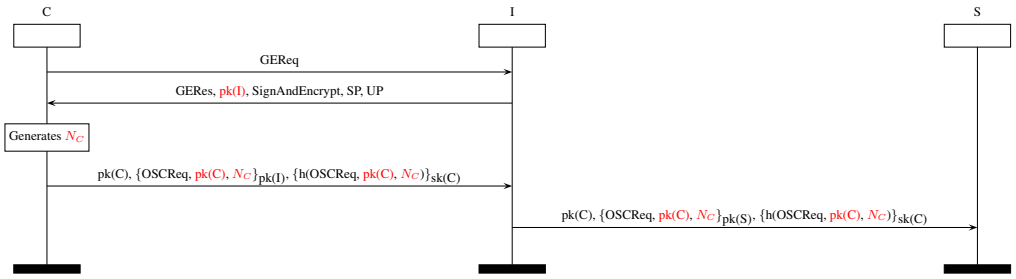


**Fig. 2.** Attack on $N_C$: $I$ usurps $C$ when speaking to $S$.

$N_C$ are replaced by $\{N_C\}_{pk(S)}$ in message 3 and all occurrences of $N_S$ in message 4 by $\{N_S\}_{pk(C)}$. Thus in security mode Sign, all the entire messages are signed but only the nonces are encrypted. More formally, message 3 and 4 of Figure 1 are replaced by:

3. $C \rightarrow S$ : $\left\{\text{OSCReq, pk(C)}, \{N_C\}_{pk(S)}, \textbf{pk(S)}\right\}_{\text{pk(S)}}, \left\{\text{h(OSCReq, pk(C)}, \{N_C\}_{pk(S)}, \textbf{pk(S)})\right\}_{\text{sk(C)}}$
4. $S \rightarrow C$ : $\left\{\text{OSCRes}, \{N_S\}_{pk(C)}, \text{ST, TTL, } \textbf{pk(C)}\right\}_{\text{pk(C)}}, \left\{\text{h(OSCRes}, \{N_S\}_{pk(C)}, \text{ST, TTL, } \textbf{pk(C)})\right\}_{\text{sk(S)}}$

We also use ProVerif to confirm the security of the protocol with all our counter-measures. The results are presented in Table 2 and show that both authentication and secrecy are now secure for security modes Sign and SignAndEncrypt. As nonces are encrypted in security mode Sign, keys remain secret.

| OPC-UA Security mode | Objectives | | | |
|---|---|---|---|---|
| | Sec $K_{CS}$ | Sec $K_{SC}$ | Auth $N_S$ | Auth $N_C$ |
| None | UNSAFE | UNSAFE | UNSAFE | UNSAFE |
| Sign | SAFE | SAFE | SAFE | SAFE |
| SignEnc | SAFE | SAFE | SAFE | SAFE |

**Table 2.** Results for fixed *OpenSecureChannel* sub-protocol

## 3 OPC-UA CreateSession

The OPC-UA*CreateSession* sub-protocol allows a client to send credentials (*e.g.* a login and a password) over an already created Secure Channel. This sub-protocol is presented in Figure 3. This protocol follows the security mode that was chosen during the *OpenSecureChannel* sub-protocol and uses the symmetric keys derived, thus encryption becomes symmetric and signature relies on a *Message Authentication Code* (MAC). Then messages sent by $C$ are encrypted using $K_{CS}$ (resp. signed with $KSig_{CS}$) and messages sent by $S$ are encrypted with $K_{SC}$ (resp. signed with $KSig_{SC}$). More formally, in message 1, $C$ sends a nonce as a challenge to $S$ with *CSReq* meaning *CreateSessionRequest*. In message 2, $S$ answers with $Sig_{N_C} = \{\text{pk(C)}, N_C\}_{\text{sk(S)}}$ and *CSRes* for *CreateSessionResponse*. The message $Sig_{N_C}$ is the response of $C$'s challenge and requires $S$ to sign with its private (asymmetric) key to prove that he is the same as in the *OpenSecureChannel* sub-protocol. For this particular use, the OPC-UA standard explicitly asks to add $C$'s public key to the signature (which confirms the counter-measure

given in Section 2.3). In message 3, $C$ answers $S$'s challenge with $Sig_{N_S}$ and sends his credentials to $S$ with *ASReq* for *ActivateSessionRequest*. Finally, in message 4, $S$ confirms to $C$ that the session is created with *ASRes* for *ActivateSessionResponse* and $N_{S2}$ a fresh nonce as a challenge that $C$ should use to refresh the session when it is timed-out. Again, *CSReq, CSRes, ASReq* and *ASRes* indicate the purpose of each message of the protocol.

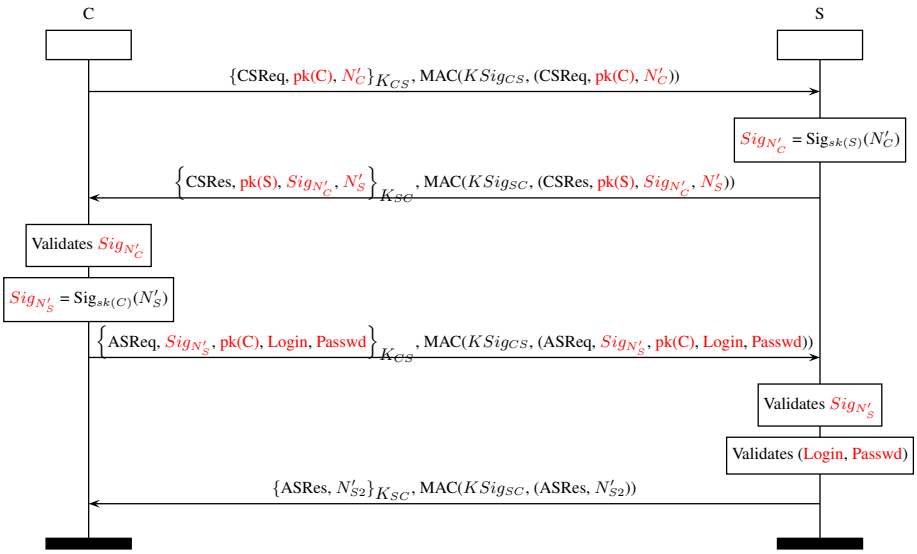


**Fig. 3.** OPC-UA*CreateSession* sub-protocol

### 3.1 Modeling

As this protocol involves logins and passwords, we assume that $C$ uses a different password for each server he speaks with. On the contrary, as mentioned in Section 2.1, we consider an intruder that can play a legitimate device that has been corrupted and would obtain the credentials of the client just by playing the protocol with him. Modeling of credentials is still not common in ProVerif. We use two functions: *Login* and *Passwd*. The first one takes as parameter the public key of a host in order to associate his login with him. This function is public for everybody. The function *Passwd* takes as parameter the private key of its owner to make it secret, but also the public key of the server to model a different password for each server. Then we provide the following equation: `verifyCreds(pk(S),Login(pk(C)),Passwd(sk(C),pk(S))) = true`. It allows the server to verify if a password and a login are matching and if the password is the one he knows (using his public key). According to our results for the *OpenSecureChannel* sub-protocol, the secrecy of the symmetric keys in security mode Sign depends on if the protocol uses key wrapping. Again, as the OPC-UA standard is not clear on how to use the mechanism in this mode, we check with and without this

security. This means that if keys are compromised, then the intruder has access to it. We consider four security objectives: (i) the secrecy of the password, (ii) the authentication of $C$ on his password, (iii) the authentication of $C$ on $Sig_{N_S}$ and (iv) the authentication of $S$ on $Sig_{N_C}$.

### 3.2 Results

Results without key wrapping (thus with keys leaked in security mode Sign, cf. Table 1) are presented in Table 3. Again, all objectives are attacked in security mode None. Also the secrecy of the password cannot hold even in security mode Sign since it will be sent by the client in plaintext during a legitimate exchange. However, both challenge-response nonces ensure authentication since the private keys are used instead of the symmetric keys. An attack on the authentication on $Passwd$ in security mode Sign is found by the tool. In this attacks the intruder replaces the credentials of $C$ by other valid credentials and recalculates the MAC of the message using the leaked keys.

| OPC-UA Security mode | Objectives | | | |
|---|---|---|---|---|
| | Sec $Passwd$ | Auth $Passwd$ | Auth $Sig_{N_S}$ | Auth $Sig_{N_C}$ |
| None | UNSAFE | UNSAFE | UNSAFE | UNSAFE |
| Sign | UNSAFE | UNSAFE | SAFE | SAFE |
| SignEnc | SAFE | SAFE | SAFE | SAFE |

**Table 3.** Results for OPC-UA*CreateSession* sub-protocol

If we consider that key wrapping is used in the *OpenSecureChannel* sub-protocol (thus without keys leaked in security mode Sign) then according to ProVerif results the authentication on $C$'s password becomes secure. This analysis shows that the use of key wrapping is crucial in security mode Sign. Thus it should be clearly said in the OPC-UA standard since missing this feature completely breaks the security of Sign mode. Moreover, $C$'s credential should also be encrypted when exchanged in Sign mode to ensure their confidentiality. Finally, we check the source code of the free implementation of OPC-UA (*FreeOpcUa*). This implementation is secure since it forces encryption of secrets even in security mode Sign.

## 4 Conclusion

We provided a formal verification of the industry standard communication protocol OPC-UA, relying its official specifications [22–25]. We used ProVerif a tool for automatic cryptographic protocol verification. Protocol modelings were tedious tasks since specifications are often elusive to allow interoperability. Particularly due to unclear statements on the use of cryptography with security mode Sign, we studied the protocol with and without counter-measures and proved the need of encryption for secrets to ensure messages security properties. We also found attacks on authentication and provided realistic counter-measures. We chose to focus on the two sub-protocols involved in the security handshake as they represent the core of the protocol's security. In the future, we aim at testing the attacks we found on official implementations which are proprietary in order to check if they filled the gap as did *FreeOpcUa*.

**Acknowledgements.** This work has been partially funded by the CNRS PEPS SISC ASSI 2016, the LabEx PERSYVAL-Lab (ANR-11-LABX-0025), the ARAMIS project (PIA P3342-146798) and "Digital trust" Chair from the University of Auvergne Foundation.

## References

1. Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51, 2011.
2. Keith Stouffer, Joe Falco, and Scarfone Karen. Guide to industrial control systems (ICS) security. *NIST special publication*, 800(82):16–16, June 2011.
3. ANSSI. Managing cybersecurity for ICS, June 2012.
4. Vinay M. Igure, Sean A. Laughter, and Ronald D. Williams. Security issues in SCADA networks. *Computers & Security*, 25(7):498 – 506, 2006.
5. Sandip C. Patel, Ganesh D. Bhatt, and James H. Graham. Improving the cyber security of SCADA communication networks. *Commun. ACM*, 52(7):139–142, July 2009.
6. Gordon R Clarke, Deon Reynders, and Edwin Wright. *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes, 2004.
7. D. Dzung, M. Naedele, T.P. von Hoff, and M. Crevatin. Security for industrial communication systems. *Proceedings of the IEEE*, 93(6):1152–1177, June 2005.
8. Qu Wanying, Wei Weimin, Zhu Surong, and Zhao Yan. The study of security issues for the industrial control systems communication protocols. In *JIMET'15*, 2015.
9. Sandip C Patel and Yingbing Yu. Analysis of SCADA security models. *International Management Review*, 3(2):68, 2007.
10. IgorNai Fovino, Andrea Carcano, Marcelo Masera, and Alberto Trombetta. Design and implementation of a secure MODBUS protocol. In *IFIP AICT'09*. 2009.
11. G. Hayes and K. El-Khatib. Securing MODBUS transactions using hash-based message authentication codes and stream transmission control protocol. In *ICCIT'13*, June 2013.
12. JH Graham and SC Patel. Correctness proofs for SCADA communication protocols. In *WM-SCI'05*, 2005.
13. David Basin, Sebastian Mödersheim, and Luca Viganò. An on-the-fly model-checker for security protocol analysis. In *ESORICS'03*, 2003.
14. Elton Saul and Andrew Hutchison. SPEAR II – the security protocol engineering and analysis resource. 1999.
15. Pascal Lafourcade and Maxime Puys. Performance evaluations of cryptographic protocols. verification tools dealing with algebraic properties. In *FPS'15*, 2015.
16. Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSF'01*, 2001.
17. D. Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, March 1981.
18. T. Dierks and E. Rescorla. The transport layer security (TLS) protocol, version 1.2. IETF RFC 5246, August 2008.
19. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS'96*, 1996.
20. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE transactions on Software Engineering*, 22(1):6, 1996.
21. Riccardo Focardi, Flaminia L Luccio, and Graham Steel. An introduction to security API analysis. In *Foundations of security analysis and design VI*, pages 35–65. Springer, 2011.

22. Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.
23. OPC Unified Architecture. Part 2: Security model, April 2013.
24. OPC Unified Architecture. Part 4: Services, August 2012.
25. OPC Unified Architecture. Part 6: Mappings, August 2012.