Formally Verifying Flow Properties in Industrial Systems*

Jannik Dreier[3], Maxime Puys[1], Marie-Laure Potet[1], Pascal Lafourcade[2] and Jean-Louis Roch[1]
[1]*Verimag, University Grenoble Alpes, Saint-Martin-d'Hères, France*
[2]*LIMOS, University Clermont Auvergne, Campus des Cézeaux, Aubière, France*
[3]*LORIA, University of Lorraine, INRIA, CNRS, France*

Keywords: Security Protocols, Industrial Systems, SCADA, Symbolic Model, Automated Verification, Flow Integrity.

Abstract: In contrast to other IT systems, industrial systems often do not only require classical properties like data confidentiality or authentication of the communication, but have special needs due to their interaction with physical world. For example, the reordering or deletion of some commands sent to a machine can cause the system to enter an unsafe state with potentially catastrophic effects. To prevent such attacks, the integrity of the message flow is necessary. We provide a formal definition of *Flow Integrity*. We apply our framework to two well-known industrial protocols: OPC-UA and MODBUS. Using TAMARIN, a cryptographic protocol verification tool, we confirm that most of the secure modes of these protocols ensure Flow Integrity given a resilient network. However, we also identify a weakness in a supposedly secure version of MODBUS.

1 INTRODUCTION

Industrial systems are often used to monitor and control a physical process such as energy production and distribution, water cleaning or transport systems. They are often simply called *Supervisory Control And Data Acquisition* (SCADA) systems. Due to their interaction with the real world, the safety of these systems is critical and any incident can potentially harm humans and the environment. Since the Stuxnet worm in 2010 (Langner, 2011), such systems increasingly face cyberattacks caused by various intruders, including terrorists or enemy governments. As the frequency of such attacks is increasing, the security of SCADA systems becomes a priority for governmental agencies, e.g. (Stouffer et al., 2011) for the NIST or (ANSSI, 2012) for the ANSSI.

While security objectives for IT systems are usually confidentiality, integrity and availability (CIA), industrial systems put a particular emphasis on integrity and availability. One property required by such systems is that all sent commands are received in the same order by the industrial machine, which is part of what we call *Flow Integrity*. This property is crucial in industrial systems since most of commands require the system to be in a specific state when they are launched. For instance, if an electric device requires to be unpowered to be manipulated, the shutdown command must arrive before any manipulation command. Inverting them could cause the device, along with its environment, to be damaged.

Automated protocol verification has been performed during the past twenty years and multiple efficient tools such as ProVerif (Blanchet, 2001), AVISPA (Armando et al., 2005), Scyther (Cremers, 2008) or TAMARIN (Meier et al., 2013) have been developed. However, they focused on cryptographic protocols for Internet such as TLS (Cremers et al., 2016) or special applications such as electronic voting (Kremer and Ryan, 2005) or auctions (Dreier et al., 2013). The Flow Integrity property differs from the properties usually verified in these classical protocols. For example, we want to ensure that messages are delivered (a liveness property), which requires a resilient channel. As Internet is not resilient, resilient channels are difficult to model in most tools that were designed to verify Internet protocols. Moreover, the order of messages is ensured in most Internet protocols as the messages have different formats, so reordering the messages simply aborts the protocol. In the context of industrial systems most of the protocols are used to transport commands, meaning that the messages always have the same format, rendering the ordering crucial. In order to ensure the correct ordering of the messages, most of the transport protocols including industrial ones use timestamps, counters and sequence

*This work has been partially funded by the CNRS PEPS SISC ASSI 2016.

Dreier, J., Puys, M., Potet, M-L., Lafourcade, P. and Roch, J-L.
Formally Verifying Flow Properties in Industrial Systems.
DOI: 10.5220/0006396500550066
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT*, pages 55-66
ISBN: 978-989-758-259-2

numbers. These solutions are notoriously difficult to model and verify using actual tools due to some theoretical limitations of the tools that often lead to non-termination. In order to face these limitations, we use the verification tool TAMARIN (Meier et al., 2013), that allows us to model counters and resilient channels that can build on previous work concerning the verification of liveness properties (Backes et al., 2017).

Contributions. To the best of our knowledge, the Flow Integrity property has not yet been formalized in the context of industrial protocols. Hence, we have two main contributions:

- We provide a formal definition of Flow Integrity in industrial control systems; a property that ensures that all messages are received without alteration, and in the same order as they were sent. We also define weaker properties, including Non-injective and Injective Message Authenticity, which ignore the ordering of messages but ensure that all received messages are unmodified (and cannot be duplicated in the injective case). We also define the corresponding Non-injective and Injective Message Delivery properties, making sure that all messages are delivered (and in the injective case the correct number of times).
- We study Flow Integrity for two real industrial protocols: MODBUS and OPC-UA. Using TAMARIN, we apply our framework to multiple versions of these protocols and discover a weakness in a version of MODBUS.

Outline. In Section 2, we discuss related work. Then in Section 3, we explain our definitions of the different properties, and in Section 4 how we modeled these properties with the TAMARIN prover. In Section 5, we apply the verification of our property to the MODBUS and OPC-UA industrial protocols. Finally, we conclude in Section 6.

2 RELATED WORK

The notion of integrity can vary a lot depending on the context. A generic definition could state that integrity is the maintenance and assurance of the accuracy and consistency of some data over its life-cycle. For instance, this notion has been applied in 1987, by Clark and Wilson in (Clark and Wilson, 1987). They proposed an access control model able to specify and analyze integrity policies. In such model, data alteration is restricted to those authorized. In 1998 in a different field, Heintze et. al. (Heintze and Riecke, 1998) analyzed the consistency of the values of variables during a program execution. Within their framework, they are able to ensure properties relying on integrity such as non-interference (i.e. the modification of a variable should not affect another). Again in a different field, in 2005, Umezawa et. al. (Umezawa and Shimizu, 2005) proposed a methodology to ensure that the description of hardware components (such as VHDL code) respects some temporal logic properties such as invalid states for state machines or invalid values for counters. Their approach relies both on model-checking and simulation.

In this paper, we studied the integrity of messages exchanged over a potentially insecure network. Traditionally, message integrity is used to detect accidental changes using error detection codes such as Cyclic Redundancy Checks (CRC). However, such detection codes do not protect against a malicious intruder since he can easily recalculate CRCs of the messages he changes. Similarly the TCP protocol protects against an accidental reordering of messages, but not against a malicious intruder that also modifies the sequence numbers used for this purpose. To guarantee message integrity in presence of malicious intruders, cryptographic primitives are needed, such as digital signatures or Message Authentication Codes (MAC).

Early works concerning the security of industrial protocols focused on discussing the security properties supported or not by protocols. In 2004, Clarke et. al. (Clarke et al., 2004) studied the security of DNP3 (*Distributed Network Protocol*) and ICCP (*Inter-Control Center Communications Protocol*). In 2005, Dzung et. al. (Dzung et al., 2005) surveyed 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, authors of the technical documentation of OPC-UA (*OPC Unified Architecture*) detailed the security measures of the protocol. In 2015, Wanying et. al. (Wanying et al., 2015) summarized the security offered by MODBUS, DNP3 and OPC-UA. None of these works give any formal proof of security properties on the protocols.

In more recent works, formal analyses started to appear for industrial protocols. In (Patel and Yu, 2007) the authors proposed a formal verification of DNP3 using OFMC (Basin et al., 2003, Open-Source Fixed-Point Model-Checker) and SPEAR II (Saul and Hutchison, 1999, Security Protocol Engineering and Analysis Resource). In (Dutertre, 2007), they detailed formal specifications of MODBUS developed using PVS, a generic theorem prover in order to help proving the consistency of an implementation with the standards. In (Fovino et al., 2009), the authors proposed a secure version of MODBUS relying on

well-known cryptographic primitives such as RSA and SHA2. In (Hayes and El-Khatib, 2013), they designed another secure version of the MODBUS protocol using hash-based message authentication codes and built on SCTP (Stream Transmission Control Protocol). In (Bratus et al., 2016), authors provided a *Deep-Packet Inspection* tool to verify syntactic correctness of DNP3 packets using the Hammer tool (Patterson and Hirsch, 2014). In (Puys et al., 2016), the authors formally verified secrecy and authentication properties of OPC-UA handshake protocols using the ProVerif tool (Blanchet, 2001). However, none of these works formally define or verify Flow Integrity.

In general – outside industrial systems – formal verification of authentication properties (Lowe, 1997) is common. As shown by (Lafourcade and Puys, 2015), this property is supported by many tools such as AVISPA (Armando et al., 2005), ProVerif (Blanchet, 2001), Scyther (Cremers, 2008) and TAMARIN (Schmidt et al., 2012). However, our definition of integrity goes beyond the usual authentication properties, as we also consider the ordering of the messages and ensure their delivery (a liveness property), which is difficult to express and verify in most of these tools. We chose TAMARIN to build on previous work (Backes et al., 2017) concerning the modeling of resilient channels and the verification of liveness properties.

3 DEFINING AUTHENTICITY, DELIVERY AND INTEGRITY

Notations. In our definitions, we talk about sequences of messages. Let S^* denote the set of sequences over a set S. For a sequence s, we write s_i for its i-th element, $|s|$ for its length, and $idx(s) = \{1, \ldots, |s|\}$ for the set of its indices. We use $[]$ to denote the empty sequence, $[s_1, \ldots, s_k]$ to denote the sequence s of length k, and $s \cdot s'$ to denote the concatenation of the sequences s and s'. We say that the sequence $[s_1 \ldots s_n]$ is a *subchain of the sequence* $[r_1 \ldots r_m]$ if there exist sequences[2] $z_0, \ldots, z_n$ such that:

$$z_0 \cdot [s_1] \cdot z_1 \cdot [s_2] \cdot \ldots \cdot [s_{k-1}] \cdot z_{n-1} \cdot [s_n] \cdot z_n = [r_1 \ldots r_m]$$

We denote by $set(S)$ the unordered set that contains only once each element of the sequence S, and by $multiset(S)$ the unordered multiset that contains the elements of S. To distinguish operations on multisets from operations on sets we use the superscript $^\sharp$: for example $\cup$ denotes set union, whereas $\cup^\sharp$ denotes multiset union. We use regular set notation $\{\cdot\}$ for sets and multisets whenever it is clear from the context whether it is a set or a multiset.

In our model, the messages consist of terms. Let Σ_{Fun} be a finite signature of functions of the set Fun and V be a set of variables, $T_{\Sigma_{Fun}}(V)$ denotes the set of terms built using functions from Σ_{Fun} and variables from V. Unlike classical cryptographic protocols, which are a finite sequences of messages, we study transport protocols that aim at transporting commands or data from a party to another, resulting in potentially infinite sequences of messages. We call the transported commands the *payload* of the message, in contrast to, e.g., protocol headers and other additional values added by the protocol. To be able to identify the payload inside a larger protocol message, we use types. We assume that this part of the message is of type D for data, and the rest of the message has other types (e.g. H for hash or S for signatures).

Definitions & Intruder Model. We suppose a set of agents that exchange messages over a network which can be (partly[3]) controlled by a Dolev-Yao intruder (Dolev and Yao, 1981). A classical Dolev-Yao intruder has access to all messages on the public network and can modify, inject, delete or delay them. He is however limited by the cryptographic primitives used: he can only decrypt a ciphertext or forge a signature if he knows the corresponding keys. This is known as the *perfect cryptography assumption*.

We define Flow Integrity for the flow of messages between two agents A and B. More precisely, we define the integrity of message *payload*, i.e., we only aim at protecting the contents of the message, as this is what is required by the applications in industrial systems. This is modeled by syntactic subterms of type D (for data) in the messages. We restrict our integrity definitions to the payload only as we can have false attacks otherwise. For example, consider a protocol that sends each message together with a signature on the message, and a random value. The message cannot be modified due to the signature, but the random value is unprotected. If we considered the random value in our definitions the protocol would not ensure any kind of integrity, although the payload actually cannot be modified.

Definition 1. *Let $S_{A,B,D}$ be the sequence that contains the subterms of type D of all messages sent by agent A to agent B, and the sequence $R_{A,B,D}$ contains the subterms of type D of all messages received by agent B from A.*

For example, given a protocol that sends the message m of type D together with its hash $h(m)$ of type H,

[2]Note that z_i can be the empty sequence.

[3]The degree of control by the intruder will depend on the type of network and the channel hypotheses.

$S_{A,B,D}$ only contains the messages, and not the hashes. Since A might not only send messages to B but also to another agent C, and B might receive messages from A and E, we define the ordered sequence of messages that A sends to B, and the ordered sequence of messages that B received from A.

Note that we understand the notions of origin and destination from the agents perspective, i.e., a message m is in $R_{A,B,D}$ if B *believes* that it came from A. Similarly, a message m that A *wanted* to send to B, but was received by C, is still in $S_{A,B,D}$.

We now define several notions of integrity, authenticity and delivery. We have three levels of integrity, authenticity and delivery, where at each level integrity is defined as the conjunction of the corresponding authenticity and delivery properties. Intuitively, the authenticity properties ensure that message have not be altered during transmission, and the delivery properties ensure that messages are not lost.

The first notion of authenticity requires that all received messages were sent by the sender, but messages can be lost or duplicated.

Property 1. *A protocol ensures* Non-Injective Message Authenticity *(NIMA) between sender A and receiver B for data D if* $\text{set}(R_{A,B,D}) \subseteq \text{set}(S_{A,B,D})$.

It does only ensure that messages are unmodified, but not that they are actually delivered. For this, we define the corresponding delivery property.

Property 2. *A protocol ensures* Non-Injective Message Delivery *(NIMD) between sender A and receiver B for data D if* $\text{set}(R_{A,B,D}) \supseteq \text{set}(S_{A,B,D})$.

Note that message delivery is difficult to achieve using an insecure asynchronous network such as Internet, but industrial systems often use special (real-time) networks with stronger channel guarantees such as *Parallel Redundancy Protocol (PRP)* and *High-availability Seamless Redundancy (HSR)* (IEC-62439, 2016). Taking the above properties together, we obtain Non-Injective Message Integrity.

Property 3. *A protocol ensures* Non-Injective Message Integrity *(NIMI) between sender A and receiver B for data D if* $\text{set}(R_{A,B,D}) = \text{set}(S_{A,B,D})$.

To ensure that messages cannot be duplicated, we have Injective Message Authenticity and Injective Message Delivery.

Property 4. *A protocol ensures* Injective Message Authenticity *(IMA) between sender A and receiver B for data D if* $\text{multiset}(R_{A,B,D}) \subseteq \text{multiset}(S_{A,B,D})$.

Property 5. *A protocol ensures* Injective Message Delivery *(IMD) between sender A and receiver B for data D if* $\text{multiset}(R_{A,B,D}) \supseteq \text{multiset}(S_{A,B,D})$.

Both properties can be verified at the same time by checking Injective Message Integrity.

Property 6. *A protocol ensures* Injective Message Integrity *(IMI) between sender A and receiver B for data D if* $\text{multiset}(R_{A,B,D}) = \text{multiset}(S_{A,B,D})$.

Again it is easy to see that a protocol ensuring Injective Message Integrity also ensures Injective Message Delivery and Injective Message Authenticity, and that vice versa a protocol ensuring Injective Message Delivery and Injective Message Authenticity also ensures Injective Message Integrity.

Injective Message Integrity ensures that all messages are delivered, and not duplicated, but they can still be reordered. This is prevented by Flow Authenticity and Flow Delivery.

Property 7. *A protocol ensures* Flow Authenticity *(FA) between sender A and receiver B for data D if* $R_{A,B,D}$ *is a subchain of* $S_{A,B,D}$.

Property 8. *A protocol ensures* Flow Delivery *(FD) between sender A and receiver B for data D if* $S_{A,B,D}$ *is a subchain of* $R_{A,B,D}$.

Both properties can be verified at the same time by checking Flow Integrity, which corresponds to the property one would like to achieve in real systems.

Property 9. *A protocol ensures* Flow Integrity *(FI) between sender A and receiver B for data D if* $S_{A,B,D} = R_{A,B,D}$.

Again it is easy to see that a protocol ensuring Flow Integrity also ensures Flow Delivery and Flow Authenticity, and that vice versa a protocol ensuring Flow Delivery and Flow Authenticity also ensures Flow Integrity.

Note that a protocol ensuring Flow Integrity also ensures Injective Message Integrity, and that a protocol ensuring Injective Message Integrity also ensures Non-Injective Message Integrity (and analogously for the authenticity and delivery properties). This is summed up in Figure 1.

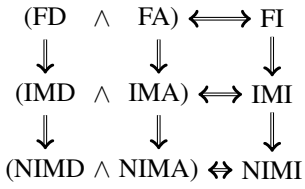

Figure 1: Relationship of our notions: $A \Rightarrow B$ if a protocol ensuring A also ensures B.

Moreover, if a protocol ensures either Flow Authenticity and Injective Message Delivery, or Flow Delivery and Injective Message Authenticity, this is sufficient to ensure Flow Integrity, as the following Theorem 1 shows.

Theorem 1. *A protocol that ensures Flow Delivery and Injective Message Authenticity also ensures Flow Integrity ($FD \wedge IMA \Rightarrow FI$). Similarly, a protocol that ensures Flow Authenticity and Injective Message Delivery, also ensures Flow Integrity ($FA \wedge IMD \Rightarrow FI$).*

Proof. Let $[s_1,\ldots,s_n] = S_{A,B,D}$ and $[r_1,\ldots,r_m] = R_{A,B,D}$. Suppose that a protocol ensures Flow Delivery and Injective Message Authenticity, i.e. we have that $S_{A,B,D}$ is a subchain of $R_{A,B,D}$, and $multiset(R_{A,B,D}) \subseteq multiset(S_{A,B,D})$. Moreover, as any protocol ensuring Flow Delivery also ensures Injective Message Delivery, we have $multiset(R_{A,B,D}) \supseteq multiset(S_{A,B,D})$, and thus $multiset(R_{A,B,D}) = multiset(S_{A,B,D})$.

This means that $n = m$, i.e. both sequences have the same length. By the definition of subchains we have that there exist sequences $z_0,\ldots,z_n$ such that $z_0 \cdot [s_1] \cdot z_1 \cdot \ldots \cdot z_{n-1} \cdot [s_n] \cdot z_n = [r_1 \ldots r_m]$. As $n = m$, we have that $[s_1,\ldots,s_n] = [r_1,\ldots,r_n]$, which is what we wanted to show.

The second proof is similar. □

4 THE TAMARIN PROVER

We now recall the syntax and semantics of labeled multiset rewriting rules, which constitute the input language of the TAMARIN prover (Schmidt et al., 2012).

4.1 Introducing the TAMARIN Prover

In TAMARIN, *equations* are used to specify properties of functions, where an equation over the signature Σ_{Fun} is an unordered pair of terms $s,t \in T_{\Sigma_{Fun}}(V)$, written $s \simeq t$. An *equational presentation* is a pair $\mathcal{E} = (\Sigma_{Fun}; E)$ of a signature Σ_{Fun} and a set of equations E. The corresponding *equational theory* $=_{\mathcal{E}}$ is the smallest Σ_{Fun}-congruence containing all instances of the equations in E. We often leave the signature Σ_{Fun} implicit and identify the equations E with the equational presentation $\mathcal{E}$. Similarly, we use $=_E$ for the equational theory $=_{\mathcal{E}}$. We say that two terms s and t are equal modulo E iff $s =_E t$. We use the subscript E to denote the usual operations on sets, sequences, and multisets where equality is modulo E instead of syntactic equality. For example, we write $\in_E$ for set membership modulo E.

Example 1. *To model MACs, let Σ_{Fun} be the signature consisting of the functions $mac(\cdot,\cdot)$ and $verify(\cdot,\cdot,\cdot)$ together with the equation*

$$verify(mac(x,k),x,k) \simeq true.$$

In TAMARIN any system is modeled with *multiset rewrite rules*. These rules manipulate multisets of *facts* which model the current state of the system, with *terms* as arguments. Formally, given a signature Σ_{Fun} and a (disjoint) set of fact symbols Σ_{Fact}, we define $\Sigma = \Sigma_{Fun} \cup \Sigma_{Fact}$, and we define the set of facts as $\mathcal{F} = \{F(t_1,\ldots,t_n) | t_i \in T_{\Sigma_{Fun}}, F \in \Sigma_{Fact} \text{ of arity } n\}$. We assume that Σ_{Fact} is partitioned into *linear* and *persistent* fact symbols; a fact $F(t_1,\ldots,t_n)$ is called linear if its function symbol F is linear, and persistent if F is persistent. Linear facts can only be consumed once, whereas persistent facts can be consumed as often as needed. In practice, messages and protocol state facts are usually modeled as linear facts, whereas the intruder knowledge or, e.g. long term keys are stored using persistent facts. Facts are said to be ground if they only contain ground terms. We denote by $\mathcal{F}^\sharp$ the set of finite multisets built using facts from $\mathcal{F}$, and by $\mathcal{G}^\sharp$ the set of multisets of ground facts.

The system's possible state transitions are modeled by *labeled multiset rewrite rules*. A labeled multiset rewrite rule is a tuple (id,l,a,r), written $id : l —[\,a\,]\!\to r$, where $l,a,r \in \mathcal{F}^\sharp$ and $id \in I$ is a unique identifier. Given a rule $ri = id : l —[\,a\,]\!\to r$, $name(ri) = id$ denotes its *name*, $prems(ri) = l$ its *premises*, $acts(ri) = a$ its *actions*, and $concs(ri) = r$ its *conclusions*. Finally, rules are said to be ground if they only contain ground facts, and $ginsts(R)$ denotes the ground instances of a set R of multiset rewrite rules, $lfacts(l)$ is the multiset of all linear facts in l, and $pfacts(l)$ is the set of all persistent facts in l. We use $mset(s)$ to highlight that s is a multiset, and we use $set(s)$ for the interpretation of s as a set, even if it is a multiset.

The semantics of a set of multiset rewrite rules P are given by a *labeled transition relation* $\to_P \subseteq \mathcal{G}^\sharp \times \mathcal{G}^\sharp \times \mathcal{G}^\sharp$, defined by the transition rule:

$$\frac{ri = id : l —[\,a\,]\!\to r \in_E ginsts(P) \quad lfacts(l) \subseteq^\sharp Spfacts(l) \subseteq S}{S \xrightarrow{set(a)}_P ((S \setminus^\sharp lfacts(l)) \cup^\sharp mset(r))}$$

Note that the initial state of a labeled transition system derived from multiset rewrite rules is the empty set of facts $\emptyset$. Each transition transforms a multiset of facts S into a new multiset of facts, according to the rewrite rule used. Moreover each transition is labeled by the actions a of the rule. These labels are used to specify security properties as explained below. Since we perform multiset rewriting modulo E, we use $\in_E$ for the rule instance. As linear facts are consumed upon rewriting, we use multiset inclusion, written $\subseteq^\sharp$, to check that all facts in $lfacts(l)$ occur sufficiently often in S. For persistent facts, we only check that each fact in $pfacts(l)$ occurs in S. To obtain the successor state, we remove the consumed linear facts and add the generated facts. The actions associated to the transition contain the set of actions of the rule instance,

the identifier of the rule, and the newly introduced variables.

Example 2. *The following multiset rewrite rules describe a simple protocol that sends messages together with a hash of the message. The first rule describes the agent A: he uses the key shared with B to send a fresh message m to B. The second rule describes B: he receives a message together with its hash. Note that the second rule can only be triggered if the input matches the premise, i.e., if the hash is correctly computed.*

$$\begin{aligned} &\mathit{Send\_Message\_A}: \\ &\quad [\mathsf{Fr}(m)] -\![\mathsf{Sent}(m)]\!\to [\mathsf{Out}((m,h(m)))], \\ &\mathit{Receive\_Message\_B}: \\ &\quad [\mathsf{In}((m,h(m)))] -\![\mathsf{Received(m)}]\!\to [] \end{aligned}$$

TAMARIN implements a Dolev-Yao intruder given by the message deduction rules *MD* below. The intruder can receive any message sent on the network, send out any term he knows, create fresh values or public values, and apply functions from the function signature. This message deduction is considered modulo the equational theory.

$$\begin{aligned} MD = \{\ &\mathsf{Out}(x) -\![]\!\to \mathsf{K}(x),\ \mathsf{K}(x) -\![\mathsf{K}(x)]\!\to \mathsf{In}(x), \\ &\mathsf{Fr}(x\colon fr) -\![]\!\to \mathsf{K}(x\colon fr),\ [] -\![]\!\to \mathsf{K}(x\colon pub)\ \} \\ \cup\{\ &\mathsf{K}(x_1),\ldots,\mathsf{K}(x_n) -\![]\!\to \mathsf{K}(f(x_1,\ldots,x_n)) \\ &\qquad \mid f \in \Sigma_{Fun} \text{ with arity } n\ \} \end{aligned}$$

Note that all messages on the public network transit via the intruder, whose rules make the connection between the Out and In facts in the protocol rules.

Moreover, in TAMARIN the Fr facts have a special semantics. These facts can only be generated using a special rule FRESH : $[] -\![\mathsf{Fr}(x)]\!\to [\mathsf{Fr}(x)]$, and each instance of the rule generates a new fresh value, as ensured by the following definition of the possible executions.

Definition 2 (Executions)**.** *Given a multiset rewriting system R we define its set of executions as*

$$\begin{aligned} \mathit{exec}^{msr}(R) = \Big\{ \emptyset \xrightarrow{A_1}_R \ldots \xrightarrow{A_n}_R S_n \mid \forall i,j \in \mathbb{N}_n, a. \\ (S_{i+1} \setminus^{\#} S_i) = \mathsf{Fr}(a)^{\sharp} \wedge \\ (S_{j+1} \setminus^{\#} S_j) = \mathsf{Fr}(a)^{\sharp} \Rightarrow i = j \Big\} \end{aligned}$$

Our security properties will be expressed as properties on the traces associated to the executions. We define the set of traces as follows.

Definition 3 (Traces)**.** *The set of traces is defined as*

$$\begin{aligned} \mathit{traces}^{msr}(R) = \Big\{ (A_1,\ldots,A_n) \mid\ \forall\, 0 \le i \le n. \\ \emptyset \stackrel{A_1}{\Longrightarrow}_R \ldots \stackrel{A_n}{\Longrightarrow}_R S_n \in \mathit{exec}^{msr}(R) \Big\} \end{aligned}$$

where $\stackrel{A}{\Longrightarrow}_R$ *is defined as* $\xrightarrow{\emptyset}{}^*_R \xrightarrow{A}_R \xrightarrow{\emptyset}{}^*_R$ *for* $A \neq \emptyset$.

In TAMARIN, security properties are specified in an expressive two-sorted first-order logic over the actions on the traces. In this logic, the sort *time* is used for time points, and $\mathcal{V}_{time}$ are the temporal variables. The other type *msg* for message is used for messages and cryptographic terms.

Definition 4 (Trace formulas)**.** *A trace atom is either false* $\bot$*, a term equality* $t_1 \approx t_2$*, a timepoint ordering* $i \lessdot j$*, a timepoint equality* $i \doteq j$*, or an action* $F@i$ *for a fact* $F \in \mathcal{F}$ *and a timepoint i. A trace formula is a first-order formula over trace atoms.*

These trace formulas are used to specify the desired security properties, and TAMARIN can then be used to check whether all traces respect a property, or whether there is an execution that violates a property.

Example 3. *Consider the multiset rewrite rules given in Example 2. The following property specifies that any message received by B was previously sent by A:*

$$\begin{aligned} &\forall i : \mathit{time}, m : \mathit{msg}. \\ &\quad \mathit{Received}(m)@i \Rightarrow (\exists j. \mathit{Sent}(m)@j \wedge j \lessdot i) \end{aligned}$$

For the formal definition of the semantics, see (Schmidt et al., 2012).

4.2 Defining our Security Properties

Using trace formulas we can specify all our properties in TAMARIN as follows. To make messages visible on the trace, we instrument the protocol rules in TAMARIN with two actions, $\mathsf{Sent}(A,B,m)$ and $\mathsf{Received}(A,B,m)$, where the first one denotes that the message m was sent by A to B, and $\mathsf{Received}(A,B,m)$ denotes that B received message m from A. Note that here we only use the message payload, i.e. m is the part of the protocol message that is of type D. Using these actions, we can define Non-Injective Message Authenticity in TAMARIN as follows.

Property 10. *A* TAMARIN *protocol model ensures* Non-Injective Message Authenticity *(NIMA) between sender A and receiver B for data D if the following formula is satisfied on all traces:*

$$\begin{aligned} &\forall i : \mathit{time}, A, B, m : \mathit{msg}. \mathit{Received}(A,B,m)@i \\ &\qquad \Rightarrow (\exists j. \mathit{Sent}(A,B,m)@j \wedge j \lessdot i) \end{aligned}$$

This definition captures precisely the definition from Section 3: we require that any message m received by B from A, i.e. $m \in \mathit{set}(R_{A,B,D})$, is included in $\mathit{set}(S_{A,B,D})$, i.e. was sent by A to B. We can define Non-Injective Message Delivery analogously by interchanging the Sent and Received actions.

Property 11. *A* TAMARIN *protocol model ensures* Non-Injective Message Delivery *(NIMD) between*

sender A and receiver B for data D if the following formula is satisfied on all traces:

$$\forall i : time, A, B, m : msg.Sent(A,B,m)@i \Rightarrow (\exists j.Received(A,B,m)@j \wedge i \lessdot j)$$

To verify Non-Injective Message Integrity we can simply check whether both Non-Injective Message Authenticity and Non-Injective Message Delivery hold.

To ensure injectivity, we have to ensure that a message cannot be duplicated, which we express as follows.

Property 12. *A* TAMARIN *protocol model ensures* Injective Message Authenticity *(IMA) between sender A and receiver B for data D if the following formula is satisfied on all traces:*

$$\forall i : time, A, B, m : msg.Received(A,B,m)@i \Rightarrow (\exists j.Sent(A,B,m)@j \wedge j \lessdot i \wedge \neg(\exists i2 : time, A2, B2 : msg.Sent(A2,B2,m)@i2 \wedge \neg(i2 \doteq i)))$$

This ensures that any received message was previously sent, and that there is not other time point where the same message is received, thus capturing the injectivity requirement[4]. The corresponding delivery property definition is obtained easily by interchanging the Sent and Received actions, as above.

Property 13. *A* TAMARIN *protocol model ensures* Injective Message Delivery *(IMD) between sender A and receiver B for data D if the following formula is satisfied on all traces:*

$$\forall i : time, A, B, m : msg.Sent(A,B,m)@i \Rightarrow (\exists j.Received(A,B,m)@j \wedge i \lessdot j \wedge \neg(\exists i2 : time, A2, B2 : msg.Received(A2,B2,m)@i2 \wedge \neg(i2 \doteq i)))$$

To verify Injective Message Integrity, we simply check both properties at the same time.

Flow Authenticity and Flow Delivery are expressed in TAMARIN as follows: we first verify that Injective Message Authenticity or Injective Message Delivery hold, respectively, and then check whether the order of messages is preserved.

Property 14. *A* TAMARIN *protocol model ensures* Flow Authenticity *(FA) between sender A and receiver B for data D if it ensures Injective Message Authenticity and if the following formula is satisfied on all traces:*

$$\forall i, j : time, A, B, m, m_2 : msg. (Received(A,B,m)@i \wedge Received(A,B,m_2)@j \wedge i \lessdot j) \Rightarrow (\exists k, l.Sent(A,B,m)@k \wedge Sent(A,B,m_2)@l \wedge k \lessdot l)$$

[4]We use unique fresh messages on the sender side to prevent false attacks that would result from the same message being sent twice and received twice.

Property 15. *A* TAMARIN *protocol model ensures* Flow Delivery *(FD) between sender A and receiver B for data D if it ensures Injective Message Delivery and if the following formula is satisfied on all traces:*

$$\forall i, j : time, A, B, m, m_2 : msg. (Sent(A,B,m)@i \wedge Sent(A,B,m_2)@j \wedge i \lessdot j) \Rightarrow (\exists k, l.Received(A,B,m)@k \wedge Received(A,B,m_2)@l \wedge k \lessdot l)$$

Again, to verify Flow Integrity, we simply check both properties at the same time.

4.3 Resilient Channels, Counters and Timestamps

As noted above, delivery properties typically require a resilient channel as an unrestricted Dolev-Yao intruder can simply delete all messages and thus prevent any message from arriving. We can model a resilient channel in TAMARIN by adding a *restriction* that enforces that all messages are eventually delivered. A restriction is a trace formula that TAMARIN will assume true, i.e., it will discard all traces violating the restriction when trying to prove a property.

To model a resilient channel, we add a new action $\mathsf{Ch\_Sent}(m)$ to all rules that send out messages on the resilient channel, and an action $\mathsf{Ch\_Received}(m)$ to all rules that receive messages from the resilient channel. Note that here m is not only the payload of type D, but the entire protocol message, and that we do not include senders or recipients. Using these actions, we can express the fact that the channel is resilient using the following formula:

$$\forall i : time, m : msg.Ch\_Sent(m)@i \Rightarrow (\exists j.Ch\_Received(m)@j \wedge i \lessdot j)$$

Note that this restriction on the intruder's capabilities does not prevent him from delaying messages for a certain time, reordering or duplicating them, or injecting new messages. This means that even when assuming a resilient channel our security properties do not hold vacuously.

We also use restrictions to model sequence numbers and timestamps. An intuitive way of modeling sequence number in TAMARIN would be to use state facts to implement a counter using a constant (e.g. *zero*) and a function (e.g. $inc(\cdot)$). Consider a protocol that simply sends out a message together with its counter:

$$Counter\_Init : [] \text{—}[]\rightarrow [\mathsf{Counter}(zero)],$$
$$Send\_Message : [\mathsf{Counter}(n), \mathsf{Fr}(m)] \text{—}[]\rightarrow [\mathsf{Counter}(inc(n)), \mathsf{Out}((m,n))]$$

Such a model usually results in non-termination. When TAMARIN tries to prove a property, it tries to find a counterexample using a backwards-search approach. More precisely, it starts from the negation of the formula, and tries to construct a valid execution by resolving the premises of all rule instances mentioned in the formula until it either has found a counterexample or a contradiction.

When analyzing the above counter model, resolving the first premise of the *Send_Message* rule results in two cases: either the premise is the conclusion of a *Counter_Init* rule, or it results from a *Send_Message* rule itself. In that case we need to resolve the same premise again, and enter a loop.

Our solution to avoid this problem is to not model the counter explicitly, but to let the intruder choose the sequence number, while limiting his choice using a restriction. Consider the rule

$$\begin{aligned} &\textit{Send\_Message}: \\ &[\mathsf{In}(n), \mathsf{Fr}(m)] — [\ \textit{Seq\_Sent}(A,B,n)\ ] \rightarrow [\mathsf{Out}((m,n))] \end{aligned}$$

and the following restriction

$$\begin{aligned} &\forall i, j : \textit{time}, A, B, \textit{seq}_1, \textit{seq}_2 : \textit{msg}. \\ &(\textit{Seq\_Sent}(A,B,\textit{seq}_1)@i \wedge \textit{Seq\_Sent}(A,B,\textit{seq}_2)@j \\ &\qquad \wedge i \lessdot j) \Rightarrow (\exists \textit{dif}.\textit{seq}_2 \approx \textit{seq}_1 + \textit{dif}) \end{aligned}$$

where "+" is an associative and commutative infix operator provided by TAMARIN. Note that "+" does not have any other associated equations and thus does not exactly correspond to an addition of numbers. In particular we do not have a neutral element 0, so that $\textit{seq}_1 + 0 \neq \textit{seq}_1$. The restriction ensures that the term representing the sequence number in any two subsequent messages increases by including a new term *dif*, but without fixing *dif* precisely. Although this abstraction allows jumps (for example increments by 2 or more) in the sequence number which would not occur in reality, it fixes an order on the sequence numbers which is sufficient to prove the properties we are interested in, as we will see in the case studies. Finally timestamps can be modeled in the same way, which means that the intruder controls the timing, but cannot go back in time.

5 APPLICATIONS TO SCADA PROTOCOLS

We verify the security of multiple variants of two industrial communication protocols (namely MODBUS and OPC-UA) to check if they guarantee the properties we defined in Section 3. The TAMARIN code is available online[5], all verifications where completed on a standard laptop within a few minutes. As mentioned earlier, all protocols presented in this Section are transport protocols which carry a request from a client to a server (the responses from the server to the client can be considered as another instance of the same protocol) over a potentially asynchronous and insecure network. We consider an unbounded number of sessions of the protocol, where each session is an arbitrary long sequence of messages.

[5]http://indusprotoverif.forge.imag.fr/DPPLR17.tar.gz

5.1 MODBUS

Description. MODBUS (MODBUS, 2004) is an industrial communication protocol designed by Modicon (now Schneider Electric) in 1979. It has become one of the most popular protocols in the domain and can be used either on serial bus or on TCP communication. We focus on the TCP version of the protocol, which is nowadays more popular than the serial version. In all MODBUS protocols, only the client is able to send requests to which the server answers (meaning that the server does never send a message on its own). In the TCP version, the message includes a sequence number in addition to the TCP sequence number. This number is called a *transaction identifier* and only increased by one at each client request. Some other terms are also part of the message, i.e. (i) a *protocol identifier* only used for compatibility with non-TCP versions, (ii) the length of the message and (iii) the *unit identifier* which is used to dispatch the command to actuators and sensors. Those three terms are public values that do not impact the security of the protocol. We choose to model them as single public header *ph*. A generic session of the protocol is displayed in Figure 2, where n is the transaction identifier, req_i is a request from the client and $resp_i$ is the corresponding response from the server.

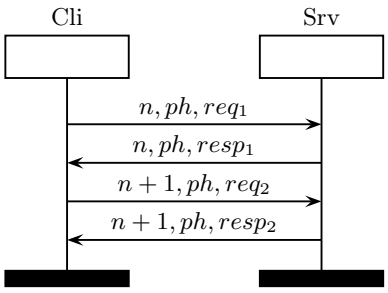

Figure 2: Two requests and responses in textbook MODBUS (MODBUS, 2004).

The protocol relies on TCP to provide countermeasures against network errors (e.g. checksums such as CRC or LRC), and does not implement any protection against malicious adversaries. Thus anyone is able to forge a fake message or modify an existing one,

allowing an adversary to run arbitrary commands on servers. To avoid such attacks, two secure versions were proposed. In (Fovino et al., 2009), the authors proposed a version of MODBUS based on well-known cryptographic primitives such as RSA and SHA2. Figure 3 presents the same session than in Figure 2 plus the counter-measures proposed in (Fovino et al., 2009) where ts_i is the timestamp of the *i*-th message.

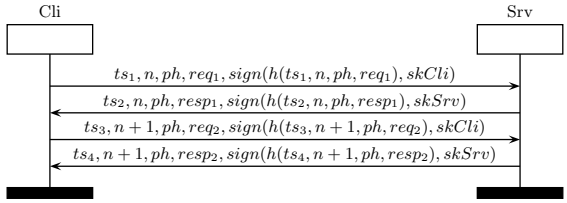

Figure 3: Two requests and responses in secure MODBUS from (Fovino et al., 2009).

In (Hayes and El-Khatib, 2013), they designed another secure MODBUS protocol based on SCTP (*Stream Control Transmission Protocol*). SCTP is a transmission layer protocol as TCP and UDP which provides protection against *Denial-of-Service* attacks. However, like TCP it provides counter-measures against network errors but none against malicious intruders. To avoid an adversary forging fake messages or modifying existing ones, Hayes et. al. added message authentication codes (MACs). Moreover, to avoid replay attacks a nonce (called *verification tag*) provided by SCTP is included in the MACs of the messages. Figure 4 details the session in Figure 2 plus the counter-measures proposed in (Hayes and El-Khatib, 2013) with *vt* the *verification tag* of the SCTP session.

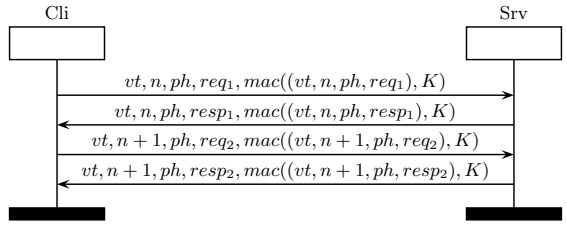

Figure 4: Two requests and responses in secure MODBUS from (Hayes and El-Khatib, 2013).

Security Analysis. We modeled the three versions of MODBUS described above and analyzed them with TAMARIN to check if they satisfy the properties we defined in Section 3. We performed a first analysis assuming an insecure network, and the results are presented in Table 1.

TAMARIN finds attacks for all properties against the standard version. This is not surprising since this version of the protocol was not intended to provide any security. However, the version with digital public key signatures from (Fovino et al., 2009) is subject to attacks since the identity of the receiver of the message is not specified in the signature. Thus an intruder is able to reroute a message to different recipient which accepts the message, violating all of our properties. This attack could be prevented by adding the receiver inside the signature, or using a different public and private key pair for each connection, which however would be equivalent to using a symmetric authentication technique such as MACs, which is done in version with MACs from (Hayes and El-Khatib, 2013). In this version the attack is prevented since the symmetric authentication keys are restricted to a specific session between a specific client and a specific server. Thus if an intruder changed the destination of a message, the new recipient would not be able to verify the MAC. This version of the protocol ensures all authenticity properties, however it still fails on all delivery properties as the intruder can simply delete all message. When assuming a resilient channel, it also ensures all delivery properties (see Table 2). Note that even when assuming a resilient channel the first two variants do still not ensure any property as messages are still not guaranteed to be delivered at the right recipient, as in the above attack.

5.2 OPC-UA

Description. OPC-UA is one of the most recent industrial communication protocols, being released in 2006 (OPC Unified Architecture, 2012). It is developed by the OPC Foundation[6], and is often referred to as the next industrial communication standard. It is a multi-level protocol, including transport and session layers. The security layer implements key agreement through a handshake. Then the client is invited to provide an authentication method such as a password or a certificate using the generated key. The transport layer consists in sending messages from the client to the server using the security keys negotiated. A generic session of the protocol is displayed in Figure 5 where:

- *mh* is a *message header* containing public values.
- *sh* is a *security header* consisting of a fresh nonce called *security token*.
- *n* is a *sequence number* incremented for each request *and* response.
- rID_i is the ID of the request to correctly associate responses.
- req_i (resp. $resp_i$) is the content of the request (resp. response).
- *pad* is a padding if needed.
- $mac(...)$ is a signature of everything above.

[6]A consortium of the main stakeholders of the domain.

Table 1: Results for MODBUS assuming an insecure network.

Protocol	NIMA	IMA	FA	NIMD	IMD	FD
Standard MODBUS (MODBUS, 2004)	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
MODBUS Sign (Fovino et al., 2009)	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
MODBUS MAC (Hayes and El-Khatib, 2013)	SAFE	SAFE	SAFE	UNSAFE	UNSAFE	UNSAFE

Table 2: Results for MODBUS assuming an resilient channel.

Protocol	NIMA	IMA	FA	NIMD	IMD	FD
Standard MODBUS (MODBUS, 2004)	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
MODBUS Sign (Fovino et al., 2009)	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
MODBUS MAC (Hayes and El-Khatib, 2013)	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE

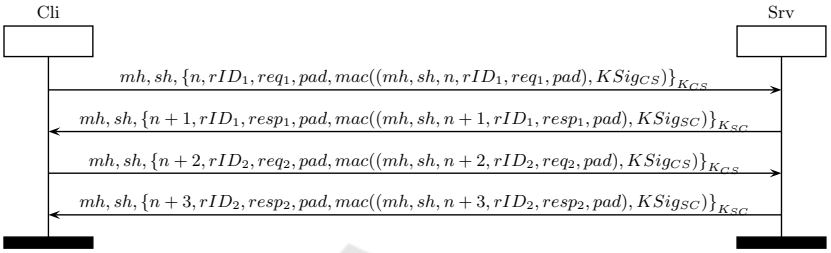

Figure 5: Two requests and responses in OPC-UA.

Only the *sequence number*, message body and signatures sent encrypted.

Finally, three security modes exist in OPC-UA:

- *SignAndEncrypt* (Figure 5): messages are signed $mac(m, K_{Sig_{XY}})$ and encrypted $\{m\}_{K_{XY}}$, where $mac(\cdot,\cdot)$ is a message authentication code function, K_{XY} the symmetric encryption key shared by X and Y, $K_{Sig_{XY}}$ the symmetric signature key shared by X and Y.
- *Sign*: it is the same as SignAndEncrypt but messages are only signed using $mac(m, K_{Sig_{XY}})$, and not encrypted. Thus message 1 (respectively 2, 3, and 4) of Figure 5 becomes: $mh, sh, n, rID_1, req_1, pad, mac((mh, sh, n, rID_1, req_1, pad), KSig_{CS})$
- *None*: messages are neither signed nor encrypted (mainly used for compatibility). Thus message 1 (respectively 2, 3, and 4) of Figure 5 becomes:

$$mh, sh, n, rID_1, req_1, pad$$

Security Analysis. We model the transport layer of OPC-UA presented in Figure 5 for the three security modes (None, Sign and SignAndEncrypt). Results for the case of an insecure network are presented in Table 3.

TAMARIN finds attacks on the version with security mode None. This is not surprising since this version was not intended to not provide any security. However both the Sign and SignAndEncrypt versions are safe for all authenticity properties. This means that having only the MACs added in the Sign version is enough to guarantee Flow Authenticity. To also have the corresponding delivery properties, we again need to assume a resilient channel (see Table 4).

Out of curiosity, we also checked a variant of the protocol with only symmetric encryption and no MAC (thus not an official version). It appears that we obtain the same results as for signatures. This is due to the fact that the symmetric keys are only shared by two participants and any message with its destination changed would not be readable by its new recipient.

OPC-UA in case of bounded sequence numbers. Until now we assumed sequence numbers to be unbounded integers from $\mathbb{N}$. However, in reality machine integers are obviously bounded and this can have an impact on properties such as Flow Integrity. To evaluate this impact, we tested a modeling of OPC-UA SignAndEncrypt (described in Figure 5) with explicitly bounded sequence numbers (in our example we bound it to four). This means that if a client sends four messages, then the fourth message has the same sequence number as the first one.

We checked the properties described in Section 3 on this version with TAMARIN and obtained the results presented in Table 5: it turns out that Flow Integrity is no longer verified. The attack works as follows: the client sends out four messages, thus the fourth message has the same sequence number as the first one. The intruder delays the first three messages so that the first message received by the server is the forth with sequence number zero. He then transmits the second message which has a sequence number of one, and is thus accepted by the server although it was actually sent earlier than the message he accepted previously.

Table 3: Results for OPC-UA (OPC Unified Architecture, 2012), assuming an insecure network.

Protocol	NIMA	IMA	FA	NIMD	IMD	FD
OPC-UA None	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
OPC-UA Sign	SAFE	SAFE	SAFE	UNSAFE	UNSAFE	UNSAFE
OPC-UA SignAndEncrypt	SAFE	SAFE	SAFE	UNSAFE	UNSAFE	UNSAFE

Table 4: Results for OPC-UA (OPC Unified Architecture, 2012), assuming a resilient channel.

Protocol	NIMA	IMA	FA	NIMD	IMD	FD
OPC-UA None	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
OPC-UA Sign	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE
OPC-UA SignAndEncrypt	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE

Table 5: Results for OPC-UA with bounded counters.

Protocol	NIMA	IMA	FA	NIMD	IMD	FD
OPC-UA SignAndEncrypt with bounded numbers Insecure Channel	SAFE	SAFE	UNSAFE	UNSAFE	UNSAFE	UNSAFE
OPC-UA SignAndEncrypt with bounded numbers Secure Channel	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE

Interestingly the described attack disappears if we assume a resilient channel. The server will accept each sequence number only once[7], and if we have two messages with the same sequence number this leads to a contradiction since both of them have to be received. This however implies that each sequence number can be used only once also on the client side, thus there can be only a finite number of messages, bounded by the range of the sequence numbers.

This analysis illustrates the need for a big range of sequence numbers. If more messages than the range of sequence numbers allows need to be exchanged, one has to reinitialize the session (i.e., exchange new keys) before running out of sequence numbers. This is the solution adopted by OPC-UA: in (OPC Unified Architecture, 2012, p. 36) it is stated that "A SequenceNumber may not be reused for any TokenId. The SecurityToken lifetime should be short enough to ensure that this never happens [...]". Our analysis underlines the importance of this requirement.

6 CONCLUSION

We provided a formal definition of Flow Integrity and other related properties in industrial systems. Flow Integrity ensures that all messages are received without alteration, and in the same order as they were sent. We checked Flow Integrity on multiple variants of two real industrial protocols: MODBUS and OPC-UA. Our analysis confirms that most of the secure modes of these protocol ensure Flow Integrity given a resilient network. However, we also identified a weakness in a supposedly secure version of MODBUS, due to an insufficient use of cryptography. Moreover, our analysis of bounded sequence numbers highlighted the importance of the renewal of session keys to avoid the reuse of sequence numbers. Unsurprisingly, the insecure modes of these protocols did not ensure any of our security properties. Moreover it turns out that to ensure delivery one has to assume a resilient channel, as the intruder can otherwise always block messages. At the same time, our results show that a resilient channel alone is not sufficient to ensure Flow Integrity: one still needs to use cryptography to prevent the intruder from rerouting or injecting messages.

In the future, we aim at testing this property on real implementations to see if we can reproduce the theoretical results. We would also like to study other industrial protocols such as DNP3 or IEC 61850. Finally we are interested in formalizing properties similar to Flow Integrity for protocols with encapsulation. Such protocols permit for example to transfer MODBUS packets through an OPC-UA channel. They are a real challenge for formal verification as there is few work on protocol composition, and it has turned out that verifying the composition is more complicated than verify the protocols independently.

[7]Note that allowing a sequence number to be reused leads to attacks on Injective Message Authenticity as the same message can be accepted multiple times.

REFERENCES

ANSSI (2012). Managing cybersecurity for ICS.

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P. H., Heám, P.-C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., R., M., Santiago, J., Turuani, M., Viganò, L., and Vigneron, L. (2005). The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV'05*.

Backes, M., Dreier, J., Kremer, S., and Künnemann, R. (2017). A novel approach for reasoning about liveness in cryptographic protocols and its application to fair exchange. In *EuroS&P'17*. To appear.

Basin, D., Mödersheim, S., and Viganò, L. (2003). An on-the-fly model-checker for security protocol analysis. In *ESORICS'03*.

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW'01*.

Bratus, S., Crain, A. J., Hallberg, S. M., Hirsch, D. P., Patterson, M. L., Koo, M., and Smith, S. W. (2016). Implementing a vertically hardened dnp3 control stack for power applications. In *ICSS'16*, pages 45–53.

Clark, D. D. and Wilson, D. R. (1987). A comparison of commercial and military computer security policies. In *Security and Privacy, 1987 IEEE Symposium on*, pages 184–184. IEEE.

Clarke, G. R., Reynders, D., and Wright, E. (2004). *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes.

Cremers, C. (2008). The Scyther Tool: Verification, falsification, and analysis of security protocols. In *CAV'08*.

Cremers, C., Horvat, M., Scott, S., and van der Merwe, T. (2016). Automated analysis and verification of TLS 1.3: 0-rtt, resumption and delayed authentication. In *SP'16*.

Dolev, D. and Yao, A. C. (1981). On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208.

Dreier, J., Lafourcade, P., and Lakhnech, Y. (2013). Formal verification of e-auction protocols. In *POST'13*.

Dutertre, B. (2007). Formal modeling and analysis of the MODBUS protocol. In *Critical Infrastructure Protection*, pages 189–204. Springer.

Dzung, D., Naedele, M., von Hoff, T., and Crevatin, M. (2005). Security for industrial communication systems. *Proceedings of the IEEE*, 93(6):1152–1177.

Fovino, I., Carcano, A., Masera, M., and Trombetta, A. (2009). Design and implementation of a secure MODBUS protocol. In *IFIP AICT'09*.

Hayes, G. and El-Khatib, K. (2013). Securing MODBUS transactions using hash-based message authentication codes and stream transmission control protocol. In *ICCIT'13*.

Heintze, N. and Riecke, J. G. (1998). The slam calculus: programming with secrecy and integrity. In *POPL'98*.

IEC-62439 (2016). Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR). International Electrotechnical Commission.

Kremer, S. and Ryan, M. D. (2005). Analysis of an electronic voting protocol in the applied pi-calculus. In *ESOP'05*.

Lafourcade, P. and Puys, M. (2015). Performance evaluations of cryptographic protocols. verification tools dealing with algebraic properties. In *FPS 2015*.

Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51.

Lowe, G. (1997). A hierarchy of authentication specifications. In *CSFW '97*.

Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The TAMARIN prover for the symbolic analysis of security protocols. In *CAV'13*.

MODBUS (2004). MODBUS IDA, MODBUS messaging on TCP/IP implementation guide v1.0a.

OPC Unified Architecture (2012). Part 6: Mappings.

Patel, S. C. and Yu, Y. (2007). Analysis of SCADA security models. *International Management Review*, 3(2):68.

Patterson, M. and Hirsch, D. (2014). Hammer parser generator. https://github.com/UpstandingHackers/hammer.

Puys, M., Potet, M., and Lafourcade, P. (2016). Formal analysis of security properties on the OPC-UA SCADA protocol. In *SAFECOMP'16*.

Saul, E. and Hutchison, A. (1999). SPEAR II – the security protocol engineering and analysis resource.

Schmidt, B., Meier, S., Cremers, C., and Basin, D. (2012). Automated analysis of diffie-hellman protocols and advanced security properties. In *CSF'12*.

Stouffer, K., Falco, J., and Karen, S. (2011). Guide to industrial control systems (ICS) security. *NIST special publication*, 800(82):16–16.

Umezawa, Y. and Shimizu, T. (2005). A formal verification methodology for checking data integrity. In *DATE'05*.

Wanying, Q., Weimin, W., Surong, Z., and Yan, Z. (2015). The study of security issues for the industrial control systems communication protocols. *JIMET'15*.