CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Security in Wireless Sensor Networks: A formal verification of protocols

Giann Nandi*
David Pereira*
Martín Vigil
Ricardo Moraes
Analúcia Schiaffino Morales
Gustavo Araújo

CISTER-TR-190506

Security in Wireless Sensor Networks: A formal verification of protocols

Giann Nandi, David Pereira, Martín Vigil, Ricardo Moraes, Analúcia Schiaffino Morales, Gustavo Araújo

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

https://www.cister-labs.pt

Abstract

The increase of the digitalization takingplace in various industrial domains is leading developerstowards the design and implementation of moreand more complex networked control systems (NCS)supported by Wireless Sensor Networks (WSN). Thisnaturally raises new challenges for the current WSNtechnology, namely in what concerns improved guaranteesof technical aspects such as real-time communicationstogether with safe and secure transmissions.Notably, in what concerns security aspects, severalcryptographic protocols have been proposed. Since thedesign of these protocols is usually error-prone, securitybreaches can still be exposed and maliciously exploitedunless they are rigorously analyzed and verified. Inthis paper we formally verify, using ProVerif, threecryptographic protocols used in WSN, regarding thesecurity properties of secrecy and authenticity. Thesecurity analysis performed in this paper is more robustthan the ones performed in related work. Our contributionsinvolve analyzing protocols that were modeledconsidering an unbounded number of participants andactions, and also the use of a hierarchical system toclassify the authenticity results. Our verification showsthat the three analyzed protocols guarantee secrecy,but can only provide authenticity in specific scenarios.

Security in Wireless Sensor Networks: A formal verification of protocols

Giann Spilere Nandi*, David Pereira*, Martín Vigil§, Ricardo Moraes§
Analúcia Schiaffino Morales§, and Gustavo Araújo§
*CISTER - Research Centre in Real-Time & Embedded Computing Systems– Portugal
§Universidade Federal de Santa Catarina – Brazil
Email: {giann,drp}@isep.ipp.pt {martin.vigil, ricardo.moraes, analucia.morales, gustavo.araujo}@ufsc.br

***Abstract*—The increase of the digitalization taking place in various industrial domains is leading developers towards the design and implementation of more and more complex networked control systems (NCS) supported by Wireless Sensor Networks (WSN). This naturally raises new challenges for the current WSN technology, namely in what concerns improved guarantees of technical aspects such as real-time communications together with safe and secure transmissions. Notably, in what concerns security aspects, several cryptographic protocols have been proposed. Since the design of these protocols is usually error-prone, security breaches can still be exposed and maliciously exploited unless they are rigorously analyzed and verified. In this paper we formally verify, using ProVerif, three cryptographic protocols used in WSN, regarding the security properties of secrecy and authenticity. The security analysis performed in this paper is more robust than the ones performed in related work. Our contributions involve analyzing protocols that were modeled considering an unbounded number of participants and actions, and also the use of a hierarchical system to classify the authenticity results. Our verification shows that the three analyzed protocols guarantee secrecy, but can only provide authenticity in specific scenarios.**

***Index Terms*—Wireless Sensor Networks, Security, Formal Verification**

I. Introduction

The latest trends that are influencing automation technologies are the Internet of Things (IoT) and Cyber-Physical Systems (CPS). The application of these paradigms leads to the definition of the so-called Industry 4.0 concept, making it possible to create intelligent products, intelligent production and intelligent services [1].

Recent studies indicate that within a few years, there will be billions of devices connected to the Internet, forming the Web of Things (WoT), once the devices that make up the IoT also become available on the World Wide Web [2]. The effective deployment of Industry 4.0 and WoT depends, amongst other things, on the development of 5G networks, which is a generation of telecommunication networks that will combine both wired and wireless communications from providers of public and private access [3].

To deal with the diversity of wireless IoT systems, 5G technology will need to integrate different networking technologies, ensuring the same level of Quality of Service (QoS) offered by the wired technologies. In this context, several wireless technologies are being developed and it is likely that, in the near future, a widely-accepted standard emerges from current wireless networking solutions, where the protocols IEEE 802.11 and IEEE 802.15.4 are the leading candidates.

In particular, the IEEE 802.15.4 protocol is the reference standard used in many Wireless Sensor Network (WSN) technologies, such as ZigBee, WirelessHart, ISA100, etc. Currently, there is a trend towards the use of these technologies in many application areas, including smart grids, smart agriculture, structural health monitoring, and Industry 4.0. Two of the major requirements of these applications areas are the support of QoS and security [4], [5]. Regarding the latter aspect, one of the main efforts must be on providing security protocols that present proofs that they comply with the security requirements of the target applications. Security protocols describe the actions that need to be followed in order to establish, despite of the possible influence of attackers, secure data communication between the communicating devices of a network. As WSN nodes are resource limited entities, the use of traditional security protocols is not a suitable choice [6]. Consequently, WSN need protocol specifications that do not require as much computation power and energy as traditional protocols do [7].

It is known that getting security protocols to work with no flaws is something challenging. There are numerous examples of protocols that were considered to be secure until proven wrong by some kind of formal verification. The classical example is the protocol proposed by Needham and Schroeder [8], which two decades after being proposed, Lowe [9] showed to be not secure.

Achieving secure communication with security protocols is the result of successfully guaranteeing security properties [6]. Two important security properties are secrecy and authenticity. Secrecy is the ability to hide confidential information from unauthorized participants in the network [10]; authenticity is the guarantee regarding the identity of the sender of a message in a hostile environment with numerous participants

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); also by the Norte Portugal Regional Operational Programme (NORTE 2020) under the Portugal 2020 Partnership Agreement, through the European Regional Development Fund (ERDF) and also by national funds through the FCT, within project NORTE-01-0145-FEDER-028550 (REASSURE). The authors would also like to acknowledge the support from the following funding agency: CNPq-Brazil (400508/2014-1).

978-1-7281-2927-3/19/$31.00 ©2019 IEEE

[10]. Lowe [11] proposes a four-level hierarchical system to define the authenticity strength of messages exchanged by two participants, where each level provides stronger guarantees of authenticity than the previous one in the hierarchy. Basically, these levels differ on: a) how confident one participant is that another participant follows the protocol properly; b) how confident participants are that they agree on the same exchanged data; and c) whether there is an one-to-one correspondence between messages participants exchange. There are other security properties that can be verified in security protocols, but, this paper only addresses secrecy and authenticity due to the limitations of the tool used for the formal verification.

a) Contributions: We describe the efforts and techniques used to formally verify the WSN protocols presented in [12]–[14]. To the best of our knowledge, these protocols have not been formally verified with respect to secrecy and authenticity. Our approach uses ProVerif[1], an automatic cryptographic protocol verifier that is based on the Dolev-Yao model. Our verification is not bounded either by a maximum number of participants or a maximum number of actions. In other words, our results are valid independently of the size of the target WSN, as well as the number of actions that can be performed by the involved participants. Results show that the three protocols provide secrecy. In contrast, they differ in the levels of authenticity they provably guarantee.

b) Paper Organization: This paper is organized as follows. In Section 2 we provide a short overview of the use of formal methods for the verification of correctness of security protocols. In Section 3 we present our models of the protocols that were verified, together with the formal verification performed on them. In Section 4, we point to some related work and compare them with some aspects of our work. Finally, in Section 5, we present our conclusion on the work described in this paper, and point the readers to future work.

II. Formal Verification

Even though the execution flow of security protocols is usually simple, it is not an easy task to design them without any security flaw [15]. Formal verification is a methodology that applies strong mathematical foundations and techniques to identify possible problems in many different areas of application.

Among the available tools to perform formal verification of security protocols, we chose ProVerif, a tool that is based on the Symbolic model (Dolev-Yao). Symbolic models [8], [16] represent the attackers as agents capable of: 1) permeating themselves in between the communication of two participants in any process of the protocol; 2) modifying and copying fragments of information sent in the network; 3) replicating messages; 4) forging messages; 5) keeping track of all messages sent in the network; 6) actively participating as normal agents in the protocol; and 7) receiving responses sent to other participants. One of the reasons to adopt ProVerif is its successful application in previous works, notably [17]–[20].

[1]http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

ProVerif can be used for proving the security properties of secrecy and authenticity. Secrecy can be defined as an assurance that private or confidential data cannot be disclosed to unauthorized agents [21]. For ProVerif, secrecy, in practice, is achieved when an unauthorized agent cannot derive some specific information from the data that is being manipulated by the protocol [22].

Authenticity can be defined as the property of being genuine, verifiable, and trustable, that is, messages are indeed sent by whom they specify that they were sent from [21]. Authenticity can be verified through correspondence assertions, which model the relationships between events in the form of "*if an event e has been executed, then e' has been previously executed*" [23]. With the argument that such definition was not enough to accurately describe authenticity, Lowe [11] proposed an hierarchical system to describe authenticity levels. This hierarchy consists of the following levels in ascending order of importance: aliveness, weak agreement, non-injective agreement, and injective agreement[2].

The verification of protocols using ProVerif is done by the elaboration of queries. ProVerif is considered to be sound, but not complete about its results [24]. The tool is capable of providing results that fit three possible outcomes: true, false, and could not be proved.

III. Verified Cryptographic Protocols

Three security protocols for WSN were formally verified in this work. The first protocol [12], which makes use of an Identity-Based Non-Interactive Key Distribution Scheme, aims at having two nodes agreeing on a symmetric key without having to exchange any sensitive data. The second protocol [13], describes the multiple steps for a key agreement between a base station and a node. Lastly, the third protocol [14] aims at having two nodes agreeing on a symmetric key, but exchanging sensitive data during the agreement.

While [12] was chosen given its relevance in the area of WSN, [13] and [14] were chosen given the lack of details in the papers that present them. Such shortage of information leaves room for different interpretations, possibly influencing the actual implementation of the protocols in many cases, which, as a consequence, could lead to potential security issues. By formally specifying and verifying the protocols using strict models, we establish solid study cases where security aspects can be verified, but still consider design choices and assumptions, which can be found in the models to follow.

Throughout this section, the models that were developed in Applied Pi Calculus [3] for the formal verification are presented. Because of space constraints, we limit ourselves, in the document, to explain the code in Listings 1-9, which describe the functioning of each protocol, and their respective queries for the formal verification. However, the complete models[4] are also available. For further details on the description of the

[2]Please refer to Lowe's work for a detailed description of each level.

[3]https://bensmyth.com/files/Smyth10-applied-pi-calculus.pdf

[4]https://gitlab.com/gspilerenandi/formalverification-wsnprotocols

protocols, we refer the reader to their original publications. The design choices and assumptions below are valid to all three protocols. If additional aspects need to be considered, these will be explicitly presented in the text.

- External agents do not have physical access to the participants of the WSN, i.e. nodes cannot be physically adulterated or captured;
- The pre-deployment phase was successfully completed and the formal verification takes place after the nodes are already deployed in their desired locations;
- Data sent during the pre-deployment phase is not initially available to the attackers, although it can occasionally become available depending on how the protocol proceeds;
- An unbounded number of participants is considered and run independently of each other. The participants can also perform an unbounded number of actions.

These design choices and assumptions are valid for the three protocols verified further in this document, but each protocol can also present particularities, which will be explicitly presented in the text.

A. Tiny PBC

For the verification of [12], the additional design choices and assumptions that were considered are presented below.

- Authenticity can only be verified through the analysis of message exchanges, so the protocol had to be extended to a point where two nodes actually exchange data;
- The pairing between two nodes happens only once per pair of nodes, avoiding the case where the same nodes keep agreeing on the same key an unbounded number of times;
- Nodes will, when being introduced in the network, broadcast their identities in accordance with the protocol creators' assumption that nodes know their neighbors' identification. They also state that new nodes can be added anytime to the network;

1) Model Presentation and Formal verification: The model below represents the key agreement between nodes **A** and **B** and is split and explained in separate parts.

Listing 1. TinyPBC: Node A

```
1  let nodeA() =
2  new IDA: id; out(CX, IDA); insert idsA(IDA); let PA = phi(EC, IDA) in new MSGA
       ↪ : bitstring;
3  !(
4   get idsB(IDRAB: id) in (
5   get alreadyPaired(=IDA, =IDRAB) in (event eNodesAlreadyPairedA(IDA, IDRAB))
6   else ( let PRA = phi(EC, IDRAB) in
7   let KEYA = calcPairKey( calcPrivateKey( PA, S ), PRA) in
8   event eEncryptMessageA(KEYA, MSGA); out(CA,(encrypt(KEYA, MSGA), IDA, IDRAB))
       ↪ ;
9   in(CB,(EMGSRA: bitstring, IDRA2: id, IDRA3: id));
10  if ((IDRA2 = IDRAB) && (IDA = IDRA3)) then let MSGRA = decrypt (EMGSRA , KEYA
       ↪ ) in
11  if (MSGRA <> MSGA) then insert alreadyPaired(IDA, IDRAB);
12  event eADecryptsMessage(MSGRA)))
13  ).
```

The execution flow of node **A** is presented in Listing 1. The process starts with node **A** declaring its own **ID** and broadcasting it on channel **CX** (which is a channel created exclusively for letting the attacker know about **A**'s **ID**). **A** proceeds by mapping its own **ID** to a point in the elliptic curve in order to obtain its public key (**PA**). **A** also creates a message (**MSGA**) that will be encrypted and sent to **B** after the symmetric key has been calculated.

In order to model the interest of **A** in pairing with as many nodes as it wishes to, the instruction lines ***4*** to ***12*** is executed an unbounded number of times. To represent the interest of **A** wanting to agree with node **B** on a symmetric key, **A** uses the table **idsB** to get a node **ID** with which **A** has not paired yet.

Next, **A** generates the symmetric key **KEYA**, which is calculated using **PA**, **S** (master key generated by the trusted authority responsible for deploying the nodes), and **B**'s public key. **A** then encrypts **MSGA** using **KEYA**, and sends it to **B**. **A** expects to receive a message from **B** which should be encrypted with a key symmetric to **KEYA**, but calculated on **B**'s side. After receiving a message, **A** tries to decipher it using **KEYA**. If the message is successfully deciphered, **A** checks if its content is different from **MSGA** and then adds **B** to its list of paired nodes. This last check is necessary in order to avoid a simple replay attack.

Listing 2. TinyPBC: Node B

```
1  let nodeB() =
2  new IDB: id; out(CX, IDB); insert idsB(IDB); let PB = phi(EC, IDB) in new MSGB
       ↪ : bitstring;
3  !(
4   get idsA(IDRBA: id) in (
5   get alreadyPaired(=IDB, =IDRBA) in (event eNodesAlreadyPairedB(IDB, IDRBA))
6   else ( let PRB = phi(EC, IDRBA) in
7   let KEYB = calcPairKey( calcPrivateKey(PB, S), PRB) in
8   event eEncryptMessageB(KEYB, MSGB); out(CB, (encrypt(KEYB, MSGB), IDB, IDRBA
       ↪ ));
9   in(CA,(EMGSRB: bitstring, IDRB2: id, IDRB3: id));
10  if ((IDRB2 = IDRBA) && (IDRB3 = IDB)) then let MSGRB = decrypt ( EMGSRB ,
       ↪ KEYB) in
11  if (MSGRB <> MSGB) then insert alreadyPaired(IDB, IDRBA);
12  event eBDecryptsMessage(MSGRB)))
13  ).
```

The process presented in Listing 2 is analogous to what was described in Listing 1, but instead of modeling the interaction of **A** with **B**, it models the interaction of **B** with **A**.

The first aspect to be verified in this protocol is the guarantee of secrecy of the master key **S**, the symmetric keys calculated by **A** and **B**, and the two messages **MSGA** and **MSGB**. ProVerif was able to prove that these five pieces of data have their secrecy guaranteed throughout the whole execution of the protocol.

The correspondence between the events that were presented in Listings 1 and 2 was used to verify the level of authenticity that the protocol is capable of ensuring.

Listing 3. TinyPBC: Authenticity Queries 1 and 2

```
1  query qM: bitstring, qEC: g, qSHKEYB: g, qIDA: id, qIDB: id; event(
       ↪ eADecryptsMessage(qM)) ==> inj-event(eEncryptMessageB(qSHKEYB, qM))
       ↪ && qSHKEYB = calcPairKey( calcPrivateKey(phi(qEC, qIDB), S), phi(qEC
       ↪ , qIDA)) && qIDB = new IDB && qIDA = new IDA && qM = new MSGB.
2  query qM: bitstring, qEC: g, qSHKEYA: g, qIDA: id, qIDB: id; event(
       ↪ eBDecryptsMessage(qM)) ==> inj-event(eEncryptMessageA(qSHKEYA, qM))
       ↪ && qSHKEYA = calcPairKey( calcPrivateKey(phi(qEC, qIDA), S), phi(qEC
       ↪ , qIDB)) && qIDB = new IDB && qIDA = new IDA && qM = new MSGA.
```

In Listing 3, two authentication queries that are used for this protocol are presented. The first one verifies whether the message that **A** deciphers was previously encrypted by a symmetric key, which was calculated using **B**'s private key and **A**'s public key. The second verification is analogous to the first by replacing **A** by **B**.

ProVerif was able to prove that, both interactions between **A** and **B**, can achieve a non-injective level of authenticity, but could not come up with a result regarding the injective level.

However, the traces presented by ProVerif for the "cannot be proved" results, in our opinion, cannot be characterized as a type of attack, given that the tool considered that, in the vision of an attacker, the value attributed to the IDA sent by A, is different from the value attributed to the IDA sent by B. In other words, ProVerif considers that bound names[5] with the same value, but sent by different entities, could possibly lead to a security breach. In this case, this is not a valid concern.

B. Herrera & Hu

For the verification of [13], the additional design choices and assumptions that were considered are presented below. The first three items are related to the defense mechanism, which was partially described in the original paper, used to avoid replay attacks.

- Each received **nonce** by the base station is associated with the node that sent it. For the base station, reused **nonces** are not considered valid for new key requests;
- The session number generated by the base station will be attached to every message sent to the node that, theoretically, started the session;
- No two **nonces** will be generated with the same value throughout the protocol execution.
- The ACK message sent by the node to the base station is encrypted with the symmetric key received by the node. Even though there is no mention of this actually happening in the paper, this is a common practice and a reasonable assumption. If this was not the case, and a plain ACK message was sent to the base station, the protocol could terminate without correctly knowing if the node indeed agreed on the generate key.

1) Model Explanation and Formal Verification: The model below represents the key agreement between the node S1 and the **base station**. The code is split and explained in parts. Later the queries used to formally verify the model are presented.

Listing 4. Herrera & Hu: Node S1

```
let node() =
new NODEID: id; out(CPUBLIC, NODEID); new NODESKEY: skey;
insert idSkey(NODEID, NODESKEY);
!(
 new REQUESTNONCE:nonce; event eSendIDandNonce(NODEID,REQUESTNONCE);
 out(CN, (NODEID, BASEID, REQUESTNONCE)); in(CBS, (ENCRYPTEDSHAREDKEY:
       ↪ bitstring, SIGNEDMESSAGE: bitstring, SESSIONNUMBERNODE: bitstring))
       ↪ ;
 let REQUESTNONCESEEION = checkSessionNumber(SESSIONNUMBERNODE) in
 if ((REQUESTNONCESEEION = REQUESTNONCE)) then
 let KNODEBS = adec(ENCRYPTEDSHAREDKEY, NODESKEY) in
 event eDecryptSharedKey(KNODEBS, NODESKEY);
 let HASHNODE = hash(KNODEBS) in event eHashOfTheReceivedKey(HASHNODE);
 let UNSIGNEDHASHNODE = checkSign(SIGNEDMESSAGE, pk(BSSKEY)) in
 if (HASHNODE = UNSIGNEDHASHNODE) then
 event eAfterUnsigningHash(UNSIGNEDHASHNODE, REQUESTNONCE);
 let ACKENCRYPTEDNODE = shenc(ACK, KNODEBS) in
 event eNodeSendsACK(ACKENCRYPTEDNODE, NODEID, REQUESTNONCE);
 out(CN, (ACKENCRYPTEDNODE, SESSIONNUMBERNODE))
).
```

In Listing 4, the execution flow of the node S1 is described. S1 starts declaring its id, NODEID, and broadcasting it on channel CPUBLIC (which is a channel created exclusively for letting the attacker know about S1's ID on ProVerif). Its private key, NODESKEY, is also generated during this initial phase of the node's process. Both NODEID and NODESKEY are stored in table **idSkey** so the **base station** knows about these values. [6]

After that, the procedure of S1 requesting a symmetric key to the **base station** starts. S1 generates a nonce called REQUESTNONCE and sends it to the **base station**. S1 then waits for a message containing two encrypted values and a SESSIONNUMBER. S1 checks if the session number was generated from the REQUESTNONCE, and continues if positive. The first encrypted value undergoes an asymmetric decryption using S1's private key, which, if successful, will transform the result in to a hash value. The second encrypted data has its signature checked by using the **base station**'s public key, resulting in a hash value that should be the same as the one that was just previously calculated. If both values are different, the protocol fails, otherwise, the protocol was successful in delivering the symmetric key KNODEBS to S1. For the later scenario, S1 proceeds to send an ACK message (encrypted using KNODEBS) to the **base station** together with the respective SESSIONNUMBER, as a confirmation that everything went correctly.

Listing 5. Herrera & Hu: Base Station

```
let baseStation() =
in(CN, (NIDRB:id, BIDRB: id, RREQUESTNONCE: nonce));
if (BIDRB = BASEID) then get idSkey(=NIDRB, NODESKEY': skey) in (
get pairsIdsNonces(IDCHECK: id, =RREQUESTNONCE) in (event eNonceAlreadyUsed(
      ↪ RREQUESTNONCE))
 else (
  insert pairsIdsNonces(NIDRB, RREQUESTNONCE);
  event eNonceStoredInTheBaseStation(NIDRB, RREQUESTNONCE);
  new keyTag: nonce; event eCreateSharedKey(keyTag,NIDRB);
  let KBSNODE = calcSharedKey(keyTag, NIDRB) in
  event eAfterTheSharedKeyCreation(KBSNODE);
  let NODEPKEY = pk(NODESKEY') in let MSGKEY = aenc(KBSNODE,NODEPKEY) in
  event eEncryptionOfTheSharedKey(MSGKEY); let MSGHASH = hash(KBSNODE) in
  event eHashOfTheCreatedKey(MSGHASH); let MSGSIGNED = sign(MSGHASH, BSSKEY)
       ↪ in
  event eCreateTheSignedMessage(MSGSIGNED);
  let SESSIONNUMBERBASE = sessionCreation(RREQUESTNONCE) in
  out(CBS, (MSGKEY, MSGSIGNED, SESSIONNUMBERBASE));
  in(CN, (ACKENCRYPTEDBASE: bitstring, SESSIONNUMBERBASER: bitstring));
  if ((SESSIONNUMBERBASER = SESSIONNUMBERBASE)) then
  let ACKDECRYPTEDBASE = shdec(ACKENCRYPTEDBASE, KBSNODE) in
  event eBaseStationReceivesACK(ACKDECRYPTEDBASE, NIDRB, RREQUESTNONCE))
)
else
 event eIdNotFound(NIDRB).
```

The execution flow of the **base station** is described in Listing 5. The process starts by receiving an ID and a **nonce** together with an identification of which entity should be receiving this information (**base station** in this case). The **base station** then verifies if the received ID identifies one of the nodes that were deployed in the target environment. If negative, the process is halted, otherwise, for the positive case, the **base station** checks whether the **nonce** used in the request has been already sent to the **base station**. If positive, then the protocol aborts, but if no such relationship is found, the **base station** stores the pair of ID and **nonce** in its records. The **base station** proceeds to generate the symmetric key (KBSNODE) to be sent to the requesting node.

KBSNODE goes through two distinct processes. The first process encrypts KBSNODE using the requesting node's public key. The second process obtainsa hash value of KBSNODE and then signs it with the **base station**'s private key. These information are sent to the requesting node together with a SESSIONNUMBER, which was based on the **nonce** used for the key

[5]Please refer to the ProVerif's manual for more details regarding the difference of bound names and free names

[6]It should be obvious to the reader that this is an adaptation of what happens in reality for the sake of modeling the protocol in ProVerif. In reality, the **base station** is the entity that actually generates these two values and stores it in the nodes.

request. The **base station** then waits for a message containing the SESSIONNUMBER and data to be deciphered using KBSNODE. If, after deciphering the message with KBSNODE, the ACK message can be verified, then the protocol was successfully completed.

The information that should be kept confidential in this protocol are: the **base station**'s private key BSSKEY, the node S1's private key NODESKEY, and the symmetric key calculated by the **base station**, which can be found in the form of KBSNODE and KNODEBS in the code. ProVerif managed to prove that their secrecy is guaranteed.

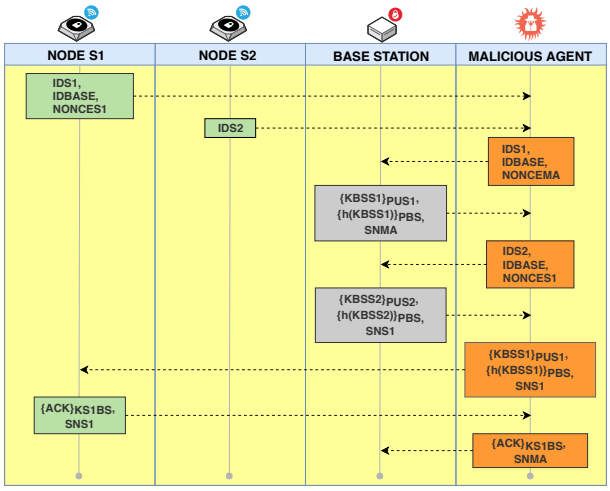

Fig. 1. ProVerif Trace (Herrera & Hu)

The authentication query presented on Listing 6, verified if the ACK message received by the **base station** was sent by node S1 as result of a successful key agreement with the **base station** after a initial key request coming from S1. ProVerif shows that there is a trace that compromises the injective and non-injective levels of authentication. For this attack to be successful, the attacker needs to learn two node IDs, and one of these nodes should to start a key request to the **base station**.

Fig. 1 illustrates such attack. The MALICIOUS AGENT, in possession of IDS1 and IDBASE, makes a key request to the **base station** acting as S1, but inserting its own generated nonce (NONCEMA). The **base station** replies to the MALICIOUS AGENT with the symmetric key encrypted with S1's public key, the signed hash of the symmetric key, and the session number based on NONCEMA. The MALICIOUS AGENT performs another key request to the base station, this time acting as S2, but using S1's nonce NONCES1. The **base station** will reply with the symmetric key encrypted with S2's public key, the signed hash of the symmetric key, and the session number based on NONCES1. The MALICIOUS AGENT forwards the encrypted symmetric, the signed hash value and the session number based on NONCES1 to S1, which then replies with the encrypted ACK message that is forwarded to the **base station** by the MALICIOUS AGENT.

Listing 6. Herrera & Hu: Correspondence Queries

```
query qKBSS1: shkey, qKEYTAG: nonce, qNODEID: id, qREQUESTNONCENODE: nonce,
      ↪ qREQUESTNONCEBS: nonce, qACKENC: bitstring; event(
      ↪ eBaseStationReceivesACK(ACK, qNODEID, qREQUESTNONCEBS))==> (inj-
      ↪ event (eNodeSendsACK(qACKENC, qNODEID, qREQUESTNONCENODE)) ==> (inj-
      ↪ event (eNonceStoredInTheBaseStation(qNODEID, qREQUESTNONCEBS)) ==>
      ↪ inj-event (eSendIDandNonce(qNODEID,qREQUESTNONCENODE)))&& qACKENC =
      ↪  shenc(ACK, qKBSS1) && qKBSS1 = calcSharedKey(qKEYTAG, qNODEID)&&
      ↪ qREQUESTNONCEBS = qREQUESTNONCENODE.
```

Considering that the **base station** and the node S1 do not agree on all the parameters of the communication during every single step of the protocol, both the injective and the non-injective level of correspondence cannot be considered. As a result, the authentication from the **base station** to the node S1 is classified as a weak agreement, given that the **base station** was not participating as a responder to S1 during its turn. The authentication of S1 to the **base station** is classified as aliveness, given that, even though S1 was participating as responder in its turn, the **base station** cannot verify the identity of S1.

C. Saqib

The model coded below represents the key agreement between two nodes, A and B, for protocol [14]. The model is split and explained in parts.

Listing 7. Najmus: Node A

```
let nodeA() =
new KA: skey; new IDA: id; let PUA = calcPublicKey(KA, GP) in
insert publicKeysA(IDA, PUA); out(CX, PUA);
!(
 get publicKeysB(IDBR1: id, PURB: pkey) in (
 get alreadyPaired(=IDA, =IDBR1) in (event eNodesAlreadyPairedA(IDA, IDBR1))
 else ( new X: number; let XG = calcSecretKey(X, GP) in
 event eNodeACreatesTheSecretKey(XG);
 let KAXPUB = encryption( addsSkeyPlusNumber(KA, X), PURB) in
 event eSendKAXPUB(KAXPUB); out(C1, (KAXPUB, IDA, IDBR1));
 in(C2, (K2:g, IDBR2: id, IDAR: id)); if ((IDAR = IDA) && (IDBR2 = IDBR1))
      ↪ then
 let GY = decryption(K2, KA, PURB) in( event eDecryptedUsingKA(GY,KA);
 let XGGY = calcSharedKey(XG, GY) in insert alreadyPaired(IDA, IDBR1);
 event eNodeAComputesSharedKey(XGGY))))
).
```

As presented in Listing 7, the execution flow of node A starts with the declaring of its private key (KA), its ID, and its public key (PUA). A then stores its pair of ID and public key in table **publicKeysA**, and also broadcasts its public key on channel CX (which is a channel created exclusively for letting the attacker know about A's public key on ProVerif). A starts an unbounded number of key agreements with as many nodes as it wishes to, as long as they have not paired with A yet. Next, A generates a random number X and calculates a shared key XG to be sent to B. After that, the sum of A's private key with X is encrypted using B's public key PUB and sent it in channel C1.

A then waits for a message to arrive, which should contain the encrypted secret of B and some kind of identification that it came from B and was addressed to A. A tries to decrypt the received message by using its own private key KA and the public key of the node that theoretically sent it, PUB. If the decryption was successful, A calculates the symmetric key XGGY, which can be later used to encrypt messages, and also adds B to the list of already paired nodes.

Listing 8. Najmus: Node B

```
let nodeB() =
new KB: skey; new IDB: id; let PUB = calcPublicKey(KB, GP) in
insert publicKeysB(IDB, PUB); out(CX, PUB);
!(
  in(C1, (K1: g, IDAR1: id, IDBR: id));
  if (IDBR = IDB) then
  get alreadyPaired(=IDB, =IDAR1) in (event eNodesAlreadyPairedB(IDB, IDAR1))
  else ( new Y: number; let YG = calcSecretKey(Y, GP) in
  event eNodeBCreatesTheSecretKey(YG); get publicKeysA(=IDAR1, PURA: pkey) in
       ↪ (
  let GX = decryption(K1,KB,PURA) in event eDecryptedUsingKB(GX,KB);
  let YGGX = calcSharedKey(GX,YG) in insert alreadyPaired(IDB, IDAR1);
  event eNodeBComputesSharedKey(YGGX);
```

```
13    let KBYPUA = encryption(addsSkeyPlusNumber(KB,Y),PURA) in
14     event eSendKBYPUA(KBYPUA); out(C2, (KBYPUA, IDB, IDAR1))))
15   ).
```

In Listing 8 the execution flow of node **B** is described. The node starts by having its private key, **ID**, and public key declared. **B** then stores the pair **ID** and public key in table **publicKeysB** and also outputs its public key on channel **CX**. Later in the process, **B** receives an encrypted key with an indication that it came from node **A** to **B**. **B** verifies if it has already paired with **A** and, if not, **B** generates a random number **Y** and calculates its secret **YG**. If the decryption is successful, **B** calculates the symmetric key to be used later with **A**, and sends the sum of its private key **KB** with the random number **Y** encrypted with **A**'s public key **PUA**. With that, **A** can now also calculate the symmetric key agreed by the two nodes.

In order to verify secrecy property of the protocol, the following data should not be disclosed by unauthorized agents: **A**'s private key (**KA**), **B**'s private key (**KB**), the random generated numbers **X** and **Y**, the secret keys calculated and deciphered by **A** and **B** (which have the form of **XG, GX, YG, GY** in the code), and the symmetric keys calculated by **A** (**XGGY**) and **B** (**YGGX**). Proverif proved that all the above values are kept confidential throughout the whole execution of the protocol.

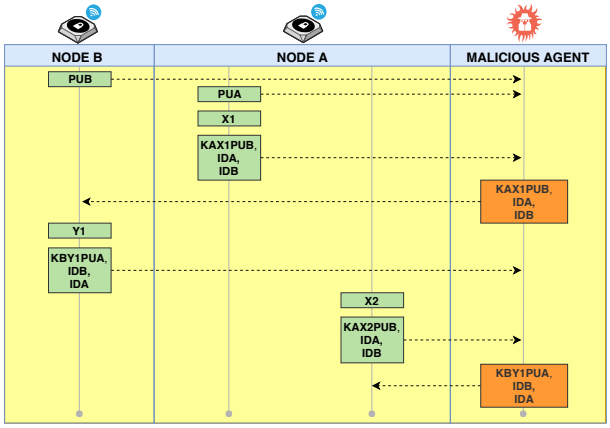

Fig. 2. Najmus Trace Second Query

Listing 9. Najmus: Authentication Queries

```
1   query qXGGY: secretkey, qYGGX: secretkey, qKBYPUA: g, qKAXPUB: g, qKB: skey,
        ↪ qKA: skey, qY: number, qX: number, qGX: secretkey, qGY: secretkey;
        ↪ event(eNodeBComputesSharedKey(qXGGY)) ==> ( inj-event (eSendKAXPUB(
        ↪ qKAXPUB)) ) && qXGGY = calcSharedKey(qGX, qGY) && qGX =
        ↪ calcSecretKey(qX,GP) && qGY = calcSecretKey(qY,GP) && qKAXPUB =
        ↪ encryption(addsSkeyPlusNumber(qKA, qX), calcPublicKey(qKB, GP)).
2   query qXGGY: secretkey, qYGGX: secretkey, qKBYPUA: g, qKAXPUB: g, qKB: skey,
        ↪ qKA: skey, qY: number, qX: number, qGX: secretkey, qGY: secretkey;
        ↪ event(eNodeAComputesSharedKey(qXGGY)) ==> ( (inj-event(
        ↪ eNodeBComputesSharedKey(qYGGX)) ==> inj-event (eSendKAXPUB(qKAXPUB))
        ↪ ) ) && qXGGY = calcSharedKey(qGX, qGY) && qYGGX = calcSharedKey(qGX,
        ↪  qGY) && qGX = calcSecretKey(qX,GP) && qGY = calcSecretKey(qY,GP) &&
        ↪  qKBYPUA = encryption(addsSkeyPlusNumber(qKB, qY), calcPublicKey(qKA
        ↪ , GP)) && qKAXPUB = encryption(addsSkeyPlusNumber(qKA, qX),
        ↪ calcPublicKey(qKB, GP)).
```

Listing 1.9 presents the queries used to verify the authenticity level guaranteed by the protocol. The first query verifies the authenticity from the message coming from **A** to **B**, by checking if the symmetric key calculated by **B** is the result of a message coming from **A**. ProVerif is successful in proving a non-injective level of correspondence, but was not able to find an answer for the injective level.

The second query tries to prove the correspondence events that precede the calculation of the symmetric key by node **A**. This query verified if **qXGGY** contains the secret key calculated by **B**, and if the symmetric key calculated by **B** contains the secret key sent by **A**. ProVerif actually finds a trace (represented by Fig. 2) in a situation where **A** tries to start a key agreement with **B**, but the response from **B** is retained by an attacker and never reaches **A**. In another attempt to establish communication, **A** tries once again to agree on a key with **B**, but receives the result from **B**'s first run of the agreement as a response. Because of that, **A** and **B** do not agree on all variables involved in the key agreement during every single step of the protocol, so the authentication from **B** to **A** can only be considered as a weak agreement. [7]

IV. Related Work and Discussion

Although there is a significant number of papers that formally verify the security of protocols, the quantity drops considerably for protocols designed specifically for WSN. For instance, [25]–[28] formally verify WSN protocols regarding authenticity and secrecy using ProVerif.

When compared to the mentioned related work, ours differs in two main points. The first is that our work, and the work presented in [28], to the best of our knowledge, are the only ones to model WSN on ProVerif where an unbounded number of agents are capable of performing an unbounded number of actions. This is especially important because the security analysis is not dependent on a specific number of individuals and actions that describe the protocol. Most of the related work describes only an unbounded number of agents, ignoring the possible effect of multiple interactions of a given agent in the network.

The second aspect distinguishing our work from others is the way that the authenticity verification is interpreted. While related works on WSN present their results with simple yes or no answers, we use the authenticity hierarchy proposed by Lowe [11] to provide a more in-depth analysis. By using such an hierarchical system, it is possible to identify more precisely the perspective that the participants of the network have regarding the authenticity aspects of the protocol.

V. Conclusions and Future Work

In this work, three security protocols for WSN were formally verified using ProVerif. With the objective of having models that were as complete as possible, an unbounded number of messages and participants were considered. This is a different and more in-depth approach when compared to similar works in the area.

The protocols were analyzed regarding the secrecy of certain crucial information and also regarding the authenticity of messages exchanged by the participants. As a result, all protocols were able to guarantee the secrecy of their sensitive data, but the results regarding the authenticity varied.

[7]Note that this attack is only valid on a scenario where **A** tries to establish a key with **B** a second time after a failed first attempt.

As future work, we plan to address the vulnerabilities our analysis revealed by proposing modifications to the vulnerable protocols. Then, we plan to formally verify the modified protocols with the help of ProVerif. In addition to that, a formal verification using a computational model approach can be carried out in order to lower the level of abstraction that the symbolic analysis inevitably implies.

REFERENCES

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, March 2017.

[2] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: A survey," *FGCS*, vol. 56, pp. 684 – 700, 2016.

[3] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A survey on 5G networks for the internet of things: Communication technologies and challenges," *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[4] V. Gazis, M. Goertz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, and F. Zeiger, "Short paper: IoT: Challenges, projects, architectures," in *18th International Conference on Intelligence in Next Generation Networks*, 2015. [Online]. Available: https://doi.org/10.1109/icin.2015.7073822

[5] I. Lee and K. Lee, "The internet of things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, jul 2015. [Online]. Available: https://doi.org/10.1016/j.bushor.2015.03.008

[6] G. Sharma, S. Bala, and A. K. Verma, "Security frameworks for wireless sensor networks-review," *Procedia Technology*, vol. 6, pp. 978–987, 2012. [Online]. Available: https://doi.org/10.1016/j.protcy.2012.10.119

[7] B. Sun, C.-C. Li, K. Wu, and Y. Xiao, "A lightweight secure protocol for wireless sensor networks," *Computer Communications*, vol. 29, no. 13, pp. 2556 – 2568, 2006, wirelsess Senson Networks and Wired/Wireless Internet Communications. [Online]. Available: https://bit.ly/2v6bX01

[8] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978. [Online]. Available: http://doi.acm.org/10.1145/359657.359659

[9] G. Lowe, "An attack on the needham-schroeder public-key authentication protocol," *Inf. Process. Lett.*, vol. 56, no. 3, pp. 131–133, Nov. 1995. [Online]. Available: http://dx.doi.org/10.1016/0020-0190(95)00144-2

[10] G. Padmavathi and D. Shanmugapriya, "A survey of attacks, security mechanisms and challenges in wireless sensor networks," *CoRR*, vol. abs/0909.0576, 2009. [Online]. Available: http://arxiv.org/abs/0909.0576

[11] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, ser. CSFW '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 31–. [Online]. Available: http://dl.acm.org/citation.cfm?id=794197.795075

[12] L. B. Oliveira, M. Scott, J. Lopez, and R. Dahab, "Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," pp. 173–180, June 2008.

[13] A. Herrera and W. Hu, "A key distribution protocol for wireless sensor networks," in *37th Annual IEEE Conference on Local Computer Networks*, Oct 2012, pp. 140–143.

[14] N. Saqib, "Key exchange protocol for WSN resilient against man in the middle attack," in *IEEE International Conference on Advances in Computer Applications (ICACA)*, Oct 2016, pp. 265–269.

[15] M. Avalle, A. Pironti, and R. Sisto, "Formal verification of security protocol implementations: a survey," *Formal Aspects of Computing*, vol. 26, no. 1, pp. 99–123, dec 2012. [Online]. Available: https://doi.org/10.1007/s00165-012-0269-9

[16] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theor.*, vol. 29, no. 2, pp. 198–208, Sep. 1983. [Online]. Available: http://dx.doi.org/10.1109/TIT.1983.1056650

[17] K. T. Nguyen, N. Oualha, and M. Laurent, *Authenticated Key Agreement Mediated by a Proxy Re-encryptor for the Internet of Things*. Cham: Springer International Publishing, 2016, pp. 339–358.

[18] L. Hirschi, D. Baelde, and S. Delaune, "A method for verifying privacy-type properties: the unbounded case," in *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P'16)*, M. Locasto, V. Shmatikov, and Ú. Erlingsson, Eds. San Jose, California, USA: IEEECSP, May 2016, pp. 564–581.

[19] J. Diaz, D. Arroyo, and F. B. Rodriguez, "On securing online registration protocols: Formal verification of a new proposal," *Knowledge-Based Systems*, vol. 59, no. Supplement C, pp. 149 – 158, 2014.

[20] P. Kleberger and G. Moulin, "Short paper: Formal verification of an authorization protocol for remote vehicle diagnostics," in *EEE Vehicular Networking Conference*, Dec 2013, pp. 202–205.

[21] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010.

[22] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016. [Online]. Available: http://dx.doi.org/10.1561/3300000004

[23] T. Y. C. Woo and S. S. Lam, "A semantic model for authentication protocols," in *Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1993, pp. 178–194.

[24] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *ProVerif 1.97pl1: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, sep 2017.

[25] M. Alshammari and K. Elleithy, "Efficient key distribution protocol for wireless sensor networks," pp. 980–985, 01 2018.

[26] Q. Jiang, S. Zeadally, J. Ma, and D. He, "Lightweight three-factor authentication and key agreement protocol for internet-integrated wireless sensor networks," *IEEE Access*, vol. 5, pp. 3376–3392, 2017.

[27] V. Cambazoglu, R. Gutkovas, J. Åman Pohjola, and B. Victor, "Modelling and analysing a wsn secure aggregation protocol : A comparison of languages and tool support," Uppsala University, Computing Science, Tech. Rep. 2015-033, 2015. [Online]. Available: http://www.it.uu.se/research/publications/reports/2015-033/

[28] Y. W. Law, G. Moniava, Z. Gong, P. Hartel, and M. Palaniswami, "KALwEN: a new practical and interoperable key management scheme for body sensor networks," *Security and Communication Networks*, vol. 4, no. 11, pp. 1309–1329, dec 2010. [Online]. Available: https://doi.org/10.1002/sec.256