Modeling Security Aspects of Network Aggregation Protocols

Frank Werner
Institut für Theoretische Informatik
Universität Karlsruhe (TH)
frank.werner@kit.edu

Raoul Steffen
Institut für Theoretische Informatik
Universität Karlsruhe (TH)
steffen@ira.uka.de

***Abstract*—We verify the correctness of a protocol for secure data aggregation using formal methods. For this purpose, a scenario is modeled and analyzed in which malicious nodes are randomly placed by an adversary among protocol compliant devices. We specify properties using linear temporal logic (LTL) and in combination with an implemented design, an exhaustive state space analysis is conducted using model checking techniques. We conclude this work with statements and assertions about the correctness of the investigated protocol and determine uncritical cases under which nodes forge packets but remain undetected.**

I. Introduction

The design of secure and safety critical protocols is even in the miniaturized world of embedded devices that run with less than 512kB of memory a challenge to engineers. It is complicated not only due to the complexity of the underlying algorithms but mainly because of the concurrent nature of such a distributed system. Simultaneous access to a common resource (like the communication medium) or the concurrent execution of tasks hamper not only implementation and design, but also the proof for correctness.

Formal methods have evolved over time to strong algorithms that can handle the complexity of those systems, and in contrary to testing and simulation, design flaws are found if they exist by an exhaustive state space analysis. One prominent tool that fulfills this claim is Spin, a model checker, that offers mechanisms for non-determinism and concurrent system modeling. The tool has demonstrated its suitability in various industrial case studies (e.g., [1], [10]) to be powerful enough in providing a sound basis for formal verification approach.

In this work we verify a protocol for authentic data aggregation by means of the Spin model checker [8] which has proven to be efficient enough to handle model instances with more than a million states and transitions. The formal modeling approach aims to complement the arguments found in the work of [5], provide new insights and possibly find potential yet undiscovered design issues in the protocol.

This paper is structured as follows. In Section II we briefly sketch the ESAWN [4] protocol with its main aspects and functionality. Section III describes the general assumptions like the adversary model. The Promela model including the modeling modalities and the properties of interest are defined and discussed in Section IV. There after in Section V the results are shown and finally Section VI wraps up this work and provides an outlook about future work.

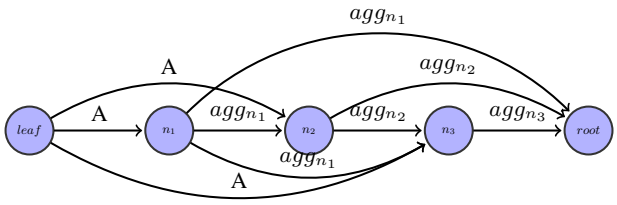


Fig. 1. ESAWN scenario of an aggregation sequence with 2 witnessing nodes ($W = 2$) i.e., that each aggregated is forwarded not only to the direct neighbor, but also to additionally two nodes which attest the correctness of the computed aggregates. For example node n_1 sends agg_{n_1} to node n_2, and in addition to n_3 and $root$ as witnesses.

II. The ESAWN Protocol

The investigated ESAWN protocol [4] (Extended secure aggregation for Wireless sensor Networks) handles the transport and aggregation of messages with guaranteed end-to-end authenticity in the presence of multiple compromised nodes.

This is mainly achieved by the use of witnesses like shown in the example of Figure 1: node $leaf$ sends value A to node n_1. Since node n_1 could be compromised witnessing nodes are involved which also receive the value A. In the above mentioned figure, two witnesses ($W = 2$) are used which attest the proper behavior of node n_1, namely node n_2 and n_3. Each witness compares the received aggregates with previously received aggregates, and in case that they equal, no faked aggregate was sent in between. On the other hand if the aggregates differ, there is at least one compromised node cheating which is made public by broadcasting an alarm message to all nodes. In the next step, a new aggregate is computed consisting of the node's own aggregate and the received ones. In the example of node n_2 the aggregation function $f_{n_2}(agg_{n_1}, A)$ computes the new aggregate agg_{n_2} which then encrypted and sent to parent nodes on the aggregation tree. By using a symmetric encryption (e.g., SKEY [2]) the authenticity of messages between nodes is achieved, since the protocol requires the authenticity of aggregates to be verifiable all the way down to the sink.

Each single data aggregate is sent and received multiple times (exactly $W+1$). Therefore it is obvious, that the protocol is causing a communication overhead and is very expensive in terms of energy. Due to this reason, the user has means to

relax the authenticity of the data that arrives at the root (base station), thus saving energy, but weakening the authenticity of the data. This is done using parameter p which gives the probability that a node verifies the correctness of an aggregate. In consequence with probability $1-p$ the packets' authenticity is not checked. The resulting trade-off behind ESAWN [4] is, the more authentic the user wants data to be sent over the network links, the more energy is needed to accomplish this.

III. Network and Intruder Assumptions

For the design of the ESAWN Model the following aspects are taken into account. We are interested in the security of the protocol and do not consider a relaxation of the authenticity. For this reason we set probability $p = 1$ meaning that all aggregates are checked. Note that in particular when setting $p < 1$ a fake aggregate will stay undetected with probability $1 - p$ and authenticity can no longer be guaranteed.

In addition, the scenario is chosen where nodes are lined up (see Fig. 1). In this case the ESAWN Protocol is not working most efficiently due to the missing aggregation of packets which result in fewer transmissions. On the other side the same properties hold here as in a tree-like scenario. Hence a tree-like spanning topology is not required to proof the protocol's correctness.

The use of aggregation is an essential part in ESAWN but does not need explicit modeling. Especially since we refrain from sending real measured data values it is sufficient to abstract from a concrete value by a data packet with what-so-ever content which dramatically reduces the model's complexity.

A. Adversary Model

The attacker can compromise some nodes (up to k) in the network by reading the nodes' memory, obtaining their secret keys, reprogramming and placing them undetected back to the network. It can randomly select the nodes although the highest potential security threat is in the case where compromised nodes are located closely together. Hereby they build a region of compromised nodes, cooperate among each other and amplify their impact on other legitimate nodes.

The *root* node is assumed to be out of the adversary's reach, operating honestly for the following reasons. If the sink would act malicious, there would be no meaningful verification of the authenticity possible since the user could not trust the base station. In addition if the attacker would have control over the root node, it has all means to take over control of the whole network sending its own requests which will not be denoted by the user.

The same holds for the *leaf* nodes. Here it can never be checked whether the extrinsically measured data is correct. As such, an adversary may forge a sensor's temperature sensor by simply using a lighter at the hardware, and the base station would not notice this counterfeit values.

IV. Promela Model

Promela (Process Meta-Language) [8] is the process description language for Spin with special emphasis on modeling process, synchronization and coordination. We model the real world ESAWN Protocol and define a variable number of N nodes to be present in our scenario. Out of this, there are up to k malicious nodes that forge packets from time to time but mostly operate normal and inconspicuous. We define variable W as the number of witnesses present, i.e., a node has to sent each packet to at least W witnesses which verify the correctness of its aggregates.

Four types of nodes form a network scenario namely the *leaf* in charge of initially sending the collected sensor data to the network which will always behave protocol conform. Among the inner nodes we distinct between *InnerNotCorrupt* nodes behaving honest and non-compromised, and *InnerCorrupt* nodes, trying to fake aggregates from time to time. The *root* node eventually received the aggregates and cannot be compromised by the adversary. The detailed behavior of the Spin process for the different type of nodes is depicted in Figure 2.

A. Modeling Channels

Before the actual Promela model of the ESAWN protocol is run, each node initializes the required message channels on the aggregation path. This means that each node obtains input channels from children nodes and is allocated outgoing channels to its succeeding parent nodes. The use of a separate channel for each node is legitimate and can be motivated by the fact that in the ESAWN protocol implementation nodes share pairwise symmetric keys (SKEY - Secure KEYing [11], [3]). In the Promela model we check these requirements by the use of channel assertions `xs` and `xr` on which nodes or processes – in terms of the Spin model – have exclusive access. The globally defined channels of type `chans`, defined as an 1-byte variable that can contain one data packet at a time.

B. State Variables

The global variables represent the overall state of the model and specify the LTL proof obligations. `CheatDetect` represents a detected faked aggregate. Note that only `InnerNotCorrupt` and `root` nodes detect a fake message since the intruder has no incentive to expose himself. This is modeling is compliant to the protocol which specifies an alarm message sent to all known surrounding nodes in case a fake aggregate is discovered.

The `root` process announces a received aggregate (`AnnounceRcv`). It then checks whether the received aggregate is correct in which case `RcvDataCorrect` is set. The variable `FakeNodes` is incremented whenever a compromised node turns active and each time a forget aggregate is sent counter `FakePacketsSnd` is incremented.

C. Process Models

Initially all four node types (`Leaf`, `InnerCorrupt`, `InnerNotCorrupt`, `Root`) initialize their channels. In

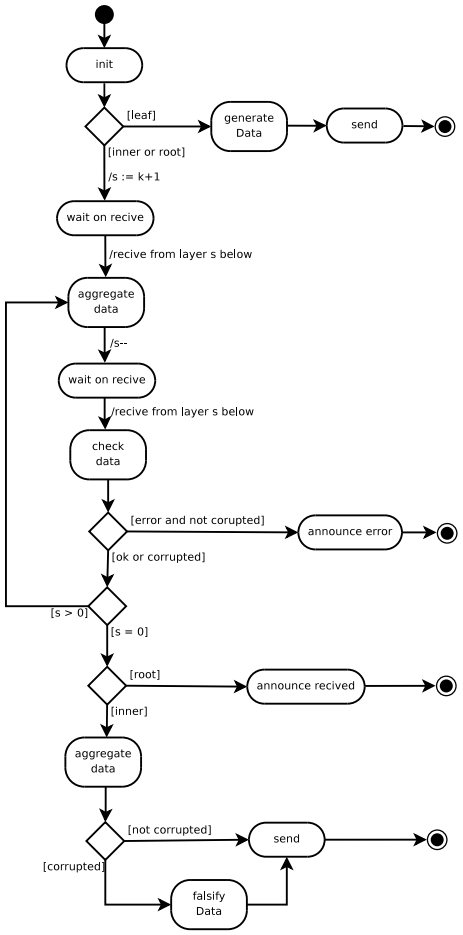


Fig. 2. Action diagram for the ESAWN protocol that is run in each Spin process independently.

more detail they set up channels with their designated predecessor and successor nodes.

1) Leaf Process: As in the scenario in Figure 1 the *leaf* generates a sample and initiates a protocol run by sending this sample its parent node and witnesses. The data packet is represented by value "0", representing the original value sent by the leaf node. After a successful sent the leaf process stops.

2) InnerNotCorrupt Node: A protocol compliant node waits until W children sent their packets and then compares the received data for equality. In case cheating is detected (`aggStore[`i`]!= aggStore[`$i-1$`]`) for one aggregate, variable `CheatDetect` is set to `true` and the node processing stops. In case that all aggregates equal, they are sent to all W parent nodes for witnessing.

3) InnerCorrupt Node: A malicious node behaves different from loyal ones. After the channel is initialized, they wait until all aggregates from child nodes are received. Where a non-malicious node verifies the aggregates, a corrupt node won't do so in order not to reveal an attack started by other corrupt nodes. In addition, a compromised node can suddenly turn active and send a forged aggregate to only a parent node, or at most W nodes.

4) Root Process: The *root node* waits for data to arrive. If aggregates arrive they are pairwise checked for equality and in case that this check fails, this is reported by setting variable `CheatDetect`. In case the results are all validated and all packets are checked for correctness, variable `RcvDataCorrect` is set to `true`. In addition variable `AnnounceRcv` is set and the process terminates.

D. Properties of Interest

In this section the model from above is feed into the model checker Spin. In the tool setting we set memory use for building the state space to 512MB. As parameters an estimated state space size of $500 \cdot 10^3$ and a maximum search depth of 10 000 steps is used. The LTL properties are defined as:

P1: $\Diamond(AnnounceRcv \vee CheatDetect)$
It is eventually the case that either the root *announces received* data ($AnnounceRcv$) or one or more nodes are cheating which is detected by at least one honest node (*cheating detected* $CheatDetect$). The use of "one node" is sufficient since it will trigger the alarm.

P2: $\Box AnnounceRcv \rightarrow (RcvDataCorrect \vee CheatDetect)$
Whenever the root node receives a data packet ($AnnounceRcv$), it is either identical with the one sent by the leaf node and no faked aggregates are sent (*received data correct* $RcvDataCorrect$) or at least one node detected a corruption ($CheatDetect$)

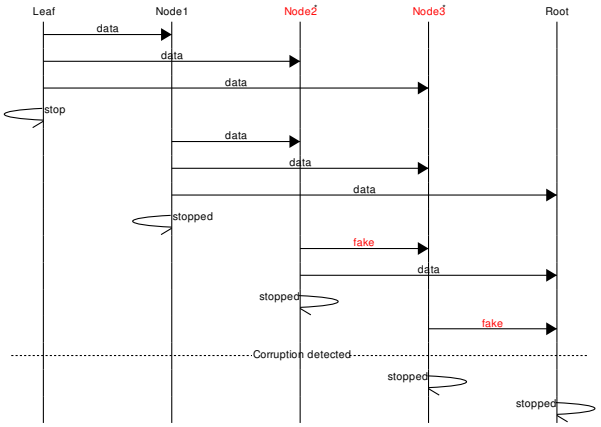
P3: $\Box(FakePacketsSnd > 0 \rightarrow \Diamond CheatDetect)$
Whenever there is a *forged packet send* ($FakePacketsSnd > 0$) this will eventually be detected and reported ($CheatDetect$)

P4: $\Box(RcvDataCorrect \rightarrow !(FakeNodes > 0))$
Whenever the data received by the root node is correct ($RcvDataCorrect$), there has been no faked message although *forging nodes* ($FakeNodes > 0$) might be present

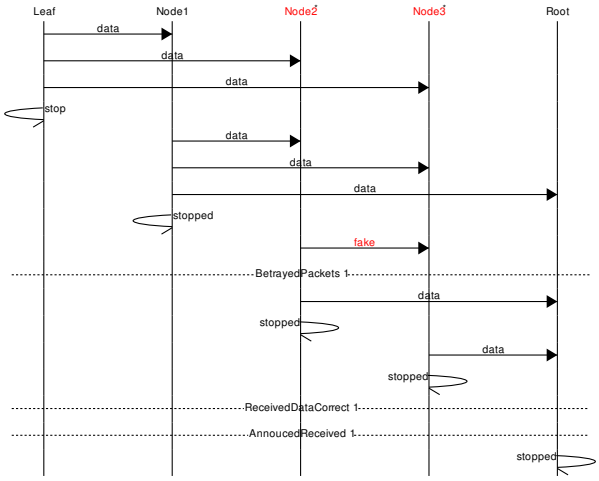
V. Results

The results are displayed for parameters $k = 2, N = 5, W = 3$ in the message sequence charts in Figure 3, which reflect only one possible trace of execution. The MSCs represent the behavior of the processes over time until termination. All global happenings like *corrupted node detected* or *betrayed packet* are displayed by dashed horizontal lines.

In the MSC of Figure 3(a) two compromised nodes (n_2, n_3) are present as denoted by the "*". The verification results are shown in Table I. Properties 1 and 2 are valid which state that no packets are lost. Consequentially, either cheating is detected, or a sound aggregate arrives at the root node. In contrary, properties 3 and 4 do not hold, proving that



(a) Correct run where cheating is detected



(b) Run where property 3 and 4 fail

Fig. 3. Message Sequence Charts (MSC) for two scenarios, where compromised nodes are indicated by the small asterisk.

property	result
$\Diamond(AnnounceRcv \vee CheatDetect)$	valid
$\Box AnnounceRcv \rightarrow (RcvDataCorrect \vee CheatDetect)$	valid
$\Box(FakePacketsSnd > 0 \rightarrow \Diamond CheatDetect)$	not valid
$\Box(RcvDataCorrect \rightarrow !(FakeNodes > 0))$	not valid

TABLE I
RESULTS WITH PARAMETERS $n = 5, k = 2, w = 3$

compromised nodes cooperate. In the MSC each of the compromised nodes is sending a faked packet to its direct child node as in Figure 3(b). Since we assume that the adversary wants to remain undetected, n_3 will not trigger an alarm since corrupt nodes cooperate. And thus the root receives two valid aggregates although a forged aggregate was sent. This does not necessary mean, that we discovered a flaw in the ESAWN protocol but rather that a scenario is possible where the data was received correctly, in the presence of a forged aggregate.

VI. CONCLUSION

The here presented analysis using the Spin tool verifies the authenticity and safety of the ESAWN protocol. In turn the failed property doesn't open space for intrusion attacks, since the adversary gained nothing in this case. Since we have not looked into the source code, we cannot guarantee the protocol's correctness after deploying it to a real world sensor network. Hence some uncertainties about the reliability of the hardware, the operation system of the sensor node, or the compiler still remain. Hence, using a more realistic model without the chosen level of abstraction and human interference would be desirable to have that could be extracted from the source code by tools like SLEDE [7] or Modex/Feaver[9]. SLEDE translates TinyOS protocols into Promela code automatically. Unfortunately only an old format of the TinyOS framework is supported by SLEDE up today.

Another way to continue this work is the generation of a behavior model using the TinyOS build-in *NULL* platform and afterwards use software verification tools like CBMC [6]. Although this approach would be restricted to sequential properties that describe the behavior of a single node, algorithmic aspects and errors introduced during the generation could be discovered before deployment.

REFERENCES

[1] Kusy B. and Abdelwahed S. FTSP Protocol Verification using SPIN. Technical Report ISIS-06-704, ISIS technical report, May 2006.

[2] Erik-Oliver Blaß, Michael Conrad, and Martina Zitterbart. A Tree-Based Approach for Secure Key Distribution in Wireless Sensor Networks. In *REALWSN – Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[3] Erik-Oliver Blaß, Michael Conrad, and Martina Zitterbart. A Tree-Based Approach for Secure Key Distribution in Wireless Sensor Networks. In *REALWSN – Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[4] Erik-Oliver Blaß, Joachim Wilke, and Martina Zitterbart. A Security–Energy Trade-Off for Authentic Aggregation in Sensor Networks. pages 135–137, Washington D.C., USA, September 2006. IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), Extended Abstract. ISBN: 1-4244-0732-X.

[5] Erik-Oliver Blaß, Joachim Wilke, and Martina Zitterbart. Relaxed Authenticity for Data Aggregation in Wireless Sensor Networks. Istanbul, Turkey, September 2008. 4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008). to appear.

[6] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A Tool for Checking ANSI-C Programs. In Kurt Jensen and Andreas Podelski, editors, *10th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[7] Youssef Hanna. Slede: lightweight verification of sensor network security protocol implementations. In *ESEC-FSE companion '07: The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, pages 591–594, New York, NY, USA, 2007. ACM.

[8] Gerard J. Holzmann. *The Spin Model Checker – Primer and Reference Manual*. Addison-Wesley, September 2003.

[9] Gerard J. Holzmann and Margaret H. Smith. Automating software feature verification. Technical report, Bell Laboratories, 2000.

[10] Tanaka Shin Ya, Sato Fumiaki, and Mizuno Tadanori. Multimedia network system. security protocol verification system based on spin. *Transactions of Information Processing Society of Japan*, 42(2):147–154, 2001.

[11] Martina Zitterbart and Erik-Oliver Blaß. An Efficient Key Establishment Scheme for Secure Aggregating Sensor Networks. In *ACM Symposium on Information, Computer and Communications Security*, pages 303–310, Taipei, Taiwan, March 2006. ISBN 1-59593-272-0.