On the Use of Automatic Tools for the Formal Analysis of IEEE 802.11 Key-Exchange Protocols

Manuel Cheminod
IEIIT-CNR
C.so Duca degli Abruzzi 24
manuel.cheminod@polito.it

Ivan Cibrario Bertolotti
IEIIT-CNR
C.so Duca degli Abruzzi 24
ivan.cibrario@polito.it

Luca Durante
IEIIT-CNR
C.so Duca degli Abruzzi 24
luca.durante@polito.it

Riccardo Sisto
Politecnico di Torino
C.so Duca degli Abruzzi 24
riccardo.sisto@polito.it

Adriano Valenzano
IEIIT-CNR
C.so Duca degli Abruzzi 24
adriano.valenzano@polito.it

Abstract

It is well known that the design and development of complex distributed systems, such as those used in modern factory automation and process control environments, can obtain significant benefits from the adoption of formal methods during the specification and verification phases. The importance of using formal techniques for verifying the design correctness is even more evident when aspects such as security and safety are considered and a class of protocols, known as "cryptographic" protocols, is taken into account. Cryptographic protocols, in fact, are becoming more and more used in industrial networks to support security-related services such as cryptographic keys exchange/distribution and authentication, due to the ever increasing use of internet/intranet-based connections and the introduction of wireless communications.

This paper reports on some experimental investigations on the formal verification of two cryptographic protocols, that are commonly used in industrial wireless 802.11 networks. Investigations are carried out by means of fully automatic and publicly available tools that are based on state-exploration techniques. The aim of our work is twofold: first we intend to offer a contribution in understanding whether or not the current prototype tools can be considered mature enough for helping the designer with the analysis of real protocols, and second we wish to develop some (preliminary) considerations on their characteristics and performance.

1 Introduction

The structure of modern industrial automation and production management systems has been changing deeply

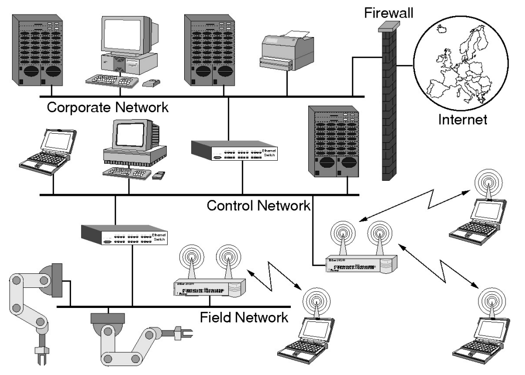

Figure 1. A Typical Topology of a Factory Network

for over a decade, being more and more pushed by the demand of connectivity for transferring information inside and outside the factory. Nowadays, in fact, several industrial distributed systems consist of a hierarchy of networks connecting intelligent devices and computers from the shop-floor to the top-floor in the typical factory scenario, as depicted in Fig. 1. At the same time, a higher degree of cooperation and openness has been progressively required for manufacturing environments, for instance in the well-known supplier-manufacturer-consumer chain, which relies heavily on the interconnection of legacy computer networks, usually by means of popular internet/intranet technologies.

Unfortunately, the interconnection of industrial systems through public networks, in general, and the internet, in particular, also makes them vulnerable to security threats that, in the past, were not considered relevant to these environments. In the last few years several security risks, that are well-known in general-purpose networks such as those deployed for office automation, have started to be considered for the industrial scenarios too, and a number of governmental agencies and industry asso-

*This work was developed in the framework of the research projects "Wireless systems for industrial automation and process control" and "Metodi e strumenti per la progettazione di sistemi software-intensive ad elevata complessità" of the Italian National Council of Research

ciations, mainly in the United States, such as CERT/CC[1], NIST[2], CICSS[3] and ISA[4], have formed technical committees and support groups concerned with the study of security attacks and related countermeasures.

In addition, it is worth remembering that, besides the risks of global connectivity, the need for enhanced security supports also depends on the introduction of new emerging technologies, such as wireless communications, that expose the industrial networks to the attacks of hackers, malicious users or even cyber-terrorists.

Many security flaws are due to incomplete or ambiguous design specifications. A number of powerful approaches, developed since the ’70s and early ’80s for granting that a system specification is able to include all the security features intended by its designers, is based on the adoption of formal methods for the specification, analysis and verification of (the most critical) parts of the system itself.

The continuous progress of research has enabled the use of formal methods for improving the design of hardware components and for analysing and verifying software systems, especially when safety-critical properties have to be taken into account, leading on to the development of several support tools that make part of the design process fully automatic.

The use of automatic or semi-automatic tools for specifying and analysing security-related properties has been investigated more recently, in particular for a class of security-critical software subsystems known as cryptographic protocols, that on the one hand play a key role in providing secure services, such as authentication or secret keys distribution, and on the other hand must be designed and implemented carefully because, even though their structure is usually quite simple, they are anyway prone to errors that are subtle and very hard to detect manually.

The focus of this paper is on the formal techniques and tools that can be used today for supporting the development of cryptographic protocols in industrial systems effectively. In particular, the paper is organised as follows: Section 2 sketches the main techniques currently being used for cryptographic protocol analysis, and the tools that today are mature and user-friendly enough to be used with a negligible effort. They are publicly available (as far as we know, no commercial ones exist) and are based on state-exploration techniques. Section 3 introduces some authentication protocols specified in the 802.11 standards [1], which have been selected as real-world cryptographic protocol examples in this paper. Specifically, our aim is to show how the security-related issues of such protocols can be studied by means of a formal approach and how the tools can both automatically analyse such protocols and give insights on found attacks. This is presented in Section 4. The examples have been chosen because it is a fact that new wireless sensors and actuators, that are based on the IEEE 802.11 technology, are appearing on the market and begin to be included in an increasing number of industrial distributed control systems. Moreover, notebooks and handheld devices equipped with 802.11-compliant interfaces are also becoming popular in the development of flexible stations that manage industrial plants without requiring a physical connection to the control network. For these reasons, a formal analysis of authentication mechanisms, such as those considered in this paper, can be useful for a better understanding of a number of security-related issues that should be considered in the design of an industrial system. Last, Section 5 draws some concluding remarks and comments on the usefulness of the analysis tools considered in this paper.

[1]http://www.cert.org/
[2]http://www.nist.gov/
[3]http://www.chemicalcybersecurity.com/
[4]http://www.isa.org/

2 Formal methods and tools for the analysis of cryptographic protocols

Cryptographic protocols can be analysed and checked for possible vulnerabilities by using various techniques, during the whole development process, starting from conceptual design down to implementation. In this paper we focus on the high-level, conceptual design phase, where the logic of message exchanges is defined independently of low-level aspects such as message encodings, cryptographic algorithms and transport channels.

Of course, security vulnerabilities may also depend on such low-level aspects, but in the initial design phases it is possible to abstract away from them, focusing on the problem of defining a conceptually secure abstract protocol and deferring the analysis of low-level protocol elements to subsequent steps. This kind of decoupling is meaningful because, although cryptographic protocols may seem quite simple when analysed from such an high-level point of view, because only a few messages are exchanged among a small number of agents, experience shows that defining attack-free protocols is quite challenging, even when low-level aspects are not considered.

According to the classification given in [2], the most common kinds of vulnerabilities and attacks are:

Freshness attacks, which occur when a message captured by the intruder in a previous protocol session is replayed, possibly as a message component, in the current protocol session.

Type-flaw attacks, involving the replacement of a message component with another message of a different type by the intruder.

Parallel session (also known as *man-in-the-middle*) attacks, in which the intruder uses messages coming from one session to synthesise malicious messages and injects them into other, parallel sessions of the protocol.

Binding attacks, exploiting the protocol's failure to establish a proper binding between a public key and its owner.

In order to isolate the conceptual protocol logic from low-level details, an idealised model of the latter is assumed. For example, for the purpose of high-level analysis, cryptography is assumed to be "perfect". This entails that the following ideal properties are assumed to hold:

an encrypted message can only be decrypted by means of the corresponding key;

the encryption key cannot be deduced from the encrypted message;

an encrypted message is sufficiently redundant so that the decryption algorithm can detect whether or not it has succeeded in its task;

the attacker cannot guess and/or forge any secret key or *nonce*[5];

cryptographic hash functions are collision-free and non-invertible.

Of course, such assumptions do not hold in the real world, but if a protocol is proved faulty when assuming perfect cryptography, it is a fortiori faulty with real cryptography. Another necessary assumption concerns capabilities which potential attackers can have. It is commonly agreed on that the *Dolev-Yao* attacker model [3] represents possible activities of malicious agents accurately. According to such a model, the attacker can:

look at, delete, reorder and replay any message sent over a public communication channel;

decrypt any encrypted message for which it has got the right key, invert invertible functions and split tuples into pieces;

generate its own nonces;

forge new messages starting from pieces of messages it already knows and possibly coming from past sessions of the protocol; it can then inject the forged messages into public communication channels.

The difficulty of designing error-free cryptographic protocols comes mainly from the difficulty of foreseeing all their possible operative scenarios, which can include concurrent executions of several protocol sessions and various different attack strategies. Formal methods can help in this respect, because they enable reasoning on complex systems, including concurrent and distributed systems, with mathematical rigour. Of course, formal methods are not a substitute for other analysis techniques, such as reviews and simulations, which deal with more concrete models, but are unable to exhaustively analyse all the system states. They have rather to be used as an additional, more sophisticated support that helps reaching higher confidence about the security of a critical system.

[5]A nonce is a random, atomic object generated by a protocol party and guaranteed to be unique with respect to all other objects used in the protocol.

Formal analysis techniques can be categorised in two main classes: deductive and state exploration methods.

Deductive methods are based on formal theories, which describe reasoning systems that can be used to prove the properties of interest. Analysing a cryptographic protocol by deductive methods means first deriving a formal theory, which represents the protocol faithfully, and then looking for a proof of one or more theorems in that theory. Those theorems formally state the protocol properties of interest such as, for example, the fact that a protocol agent receives a certain message only after its legitimate sender actually transmitted it. A violation of this property could mean that an attacker has either replayed an old messages or built a fake one.

Automatic tools can support these activities only in part. Specifically, the derivation of the formal theory starting from a formal specification of the protocol can be accomplished in a completely automatic way. Instead, the subsequent theorem proving activity can only be partially supported by tools such as theorem provers and proof checkers. Such tools do the important job of ensuring consistency and correctness of the proof and may also suggest heuristically what choices have to be done, when searching for the proof. However, finding out the proof is generally very time consuming and requires considerable ingenuity and expertise. Moreover, a conclusion might never be reached in a reasonable time: when a proof is not found, it means that either the proof does not exist (i.e. the protocol is not correct) or the proof exists but it has not been found yet. So, only when a proof is found it is possible to get to a conclusion, namely that the corresponding property holds.

State exploration methods take a quite different approach, which is more similar to simulation and testing. They work by first deriving a formal (i.e. mathematical) model of the behaviour of the system to be analysed (e.g. the cryptographic protocol) and then exploring the states of the model systematically, looking for violations of the properties of interest. State exploration can take several forms known as reachability analysis, model checking and equivalence checking. In any case, state exploration methods require modelling the protocol behaviour as a reasonably sized finite state system, which generally entails introducing simplifying assumptions that can reduce the accuracy of the analysis. Nevertheless, this form of verification has the invaluable advantages of being fully automatic and of terminating always with a final response, which can be either the absence or presence of property violations in the analysed model. In turn, this check can be performed by looking for system states that either satisfy or do not satisfy a given property. For example, a raw secrecy check on a data item may consist of ensuring that no system state exists in which the intruder gets hold of it. In case of property violations, at least a counterexample is given, showing an instance of the system behaviour where the property does not hold. The counterexample is extremely useful in understanding why the protocol is vulnerable and how it can be fixed.

One of the main challenges specific to state exploration methods for cryptographic protocols is that, in principle, the model should take into account all possible behaviours of a potential attacker, which are countless. Recently, the adoption of symbolic techniques representing (infinite) classes of different messages by means of suitable symbolic messages has partially solved this problem [4].

Specifically, to avoid the unfeasible, explicit construction of the infinite set of distinct messages that the attacker could send to protocol agents whenever they carry out an input action, this set is symbolically represented by a *variable*. Variables both provide a finite representation of the infinite set in the first place, and can subsequently be constrained to either assume or not assume specific values and syntactic forms during the course of the analysis, in order to satisfy tests and requirements posed on them by the receiving agents as they check and use them.

However, another source of infinite behaviour remains, given by the unbounded number of sessions that an attacker could exploit to perform the attack. Currently, this is typically dealt with by bounding the maximum number of concurrent protocol sessions. The direct consequence of this constraint is that attacks are not revealed, when they require a higher number of modelled sessions. However, experience has shown that all the possible attacks tend to show up in scenarios with few sessions [5]. For this reason, symbolic state exploration methods can be considered accurate enough in validating cryptographic protocols.

Because of the possibility of full automation ("push-button" operation), state exploration methods are generally considered much more attractive than deductive methods, even though the latter are in principle more powerful, having the capability to fully deal with unbounded state systems. Moreover, the domain of applicability of state exploration methods is growing, because the rapid increase in computing power and memory size leads to an equally fast growth in the size of models that can be treated. Just to give a quantitative idea, current state exploration tools can typically analyse low to medium complexity security protocols such as the Needham-Schroeder public key authentication protocol [6], at least in configurations with a couple of sessions per role, and more complex protocols with at least a single session per role: as stated above, simple configurations are almost always sufficient to show up most of the flaws.

In conclusion, even though there are still margins for improvement, it can be stated that state exploration methods can now efficiently deal not only with "toy" examples, as it happened a few years ago, but with most of the protocols that are commonly used today.

The research on tools for formal analysis of cryptographic protocols is currently very active, and several prototype tools are now available. For the reasons mentioned above, the most promising ones for practical use are those based on state exploration.

Feature	**Tool**			
	S^{3}A	**OFMC**	**STA**	**Casper**
On-the-fly analysis	[a]			
Symbolic message representation				
Infinite sessions		[b]		
Non-atomic keys			[c]	
Term type system		[d]		
Automatic synthesis of the attack				

[a] only for secrecy
[b] termination not guaranteed
[c] partial
[d] user choice

Table 1. Summary of the Features and Limitations of the Tools

In this section, the following selection of state exploration tools for cryptographic protocol analysis is presented: Casper/FDR, STA, S^{3}A, and OFMC. Table 1 summarises the main features of each tool.

2.1 Casper/FDR

One of the first state-exploration automatic tools used to analyse cryptographic protocols is based on the FDR model checker, a tool marketed by Formal Systems [7]. The FDR input consists of a description of the model to be analysed and a specification representing the desired properties. Both the model description and the specification are expressed in a machine readable dialect of the process algebra CSP [8]. FDR can generate the state spaces of both the model and the specification and can check whether the model satisfies the specification, i.e., whether the model is a refinement of the specification.

The semantics of CSP language allows to describe cryptographic protocol models and related security specifications in an accurate way, but the task of writing such CSP descriptions is quite difficult and error-prone. For this reason, a front-end called Casper [9] has been developed, which takes protocol models and security properties expressed in a simple language and translates them into CSP.

Casper models the intruder according to the Dolev-Yao model and in a way which is completely transparent to the user.

The analysis performed by FDR is a classical explicit model checking, and it is not performed on-the-fly, i.e. the checks are executed only after the whole state space has been built. FDR2, the last revision of FDR, has some built-in reductions to limit the size of the state space. No symbolic technique is used to represent messages. In order to keep the model finite, Casper limits the length of messages built by the intruder as well as the number of agents operating in parallel.

2.2 S^{3}A

S^{3}A (Spi calculus Specifications Symbolic Analyser) [4], is a fully automatic software tool for the formal analysis of cryptographic protocols that reduces the verification of secrecy and authenticity properties to checks of testing equivalence [10] between specifications. S^{3}A performs such checks automatically, by exhaustive state exploration.

The input language of S^{3}A is (a machine readable version of) the spi calculus [11], a process algebra which derives from π calculus [12], with some simplifications and the addition of cryptographic primitives. The spi calculus has two basic language elements: terms, to represent data, and processes, to represent behaviours. Terms are elements of a free term algebra and are not typed. S^{3}A fully implements such semantics and supports, in particular, non-atomic keys (sometimes used in real-world protocols) and the search for type-flaw attacks (e.g. where the attacker cheats an agent by using a nonce as a key). On the contrary, if a typed algebra were used, the set of messages an attacker can inject at a certain time would be reduced by type constraints, thus narrowing the set of examined possibilities, but making the analysis faster (less values, less states).

On the other hand, the spi calculus process operators let specify input and output operations carried out by each process, as well as operations typically performed on received data (decomposition and decryption of messages, and equality tests).

Two kinds of security properties can be specified: secrecy and authenticity. To specify secrecy, it is just necessary to specify what terms are expected to be kept secret. The secrecy concept adopted by S^{3}A is stronger than the one normally adopted by other tools, because it is based on testing equivalence. This way of expressing secrecy, besides capturing the fact that an intruder must not be able to acquire knowledge of secret M, also requires that an intruder must not be able to *infer* anything about M. In other words, this secrecy specification requires that each intruder, who knows M and M', must be unable to distinguish between two sessions where M and M' are transmitted (encrypted in some way), respectively.

With respect to authenticity, S^{3}A requires two specifications to be written: the first one is the description of the protocol and the second one is a reference specification, which is similar to the protocol specification, except for the fact that the authenticity of the messages is enforced. Testing equivalence of the two specifications implies that authenticity holds.

S^{3}A deals with the whole spi calculus, with the only exception of the replication operator, which introduces an unbounded number of processes. Leaving replication out, models are kept finite thanks to symbolic representations of messages. When checking for authenticity, S^{3}A does not work on-the-fly: first it generates the whole state space of the two specifications to be compared and then it checks for equivalence. Instead, when checking for secrecy, S^{3}A can work on-the-fly. If the testing equivalence check fails, S^{3}A is capable of synthesising the spi calculus specification of an intruder that can discriminate between the checked specifications, thus possibly leading to an attack. To limit the issue of state explosion, inherent in exhaustive state exploration methods, S^{3}A exploits state space symmetries and a limited form of partial order.

2.3 STA

STA (Symbolic Trace Analyser) [13, 14] is a model checker for cryptographic protocols relying on symbolic techniques.

Also in this case, protocols are described by means of a dialect of the spi calculus [11], but a single public channel over which data are exchanged is assumed. In the underlying theory of STA, terms are untyped and can be arbitrarily nested, but all keys must be atomic, although the implementation of STA provides limited support for non-atomic keys.

The intruder is modelled implicitly and conforms to the Dolev-Yao model, with the additional ability (that has to be specified explicitly) for the intruder to assume the role of a legitimate protocol participant.

STA allows to express and verify authentication properties based on correspondence assertions: in any protocol trace a certain action β must follow an action α in the same trace. Moreover, secrecy properties about certain values are verified by means of ad-hoc actions in the specification, designed so as to check that the intruder does not learn secrets at any interaction point between the intruder and the protocol.

Roles must be finite in number and behaviour. On the other hand, the symbolic representation of terms enables to replace the infinite set of messages the intruder can send to legitimate participants on each input action of the protocol with a finite one.

The analysis stops when all possibilities have been checked without finding any attack, or when a violation is detected. In the latter case, protocol steps leading to the flaw are then shown.

2.4 OFMC

OFMC (On-the-Fly Model Checker) [15] is a model-checker for cryptographic protocols relying on *lazy* techniques, exposed below, to reduce the computational effort required to carry out the analysis.

Protocols are described by means of HLPSL (High-Level Protocol Specification Language) that, in an untyped free-term algebra context, supports both symmetric and asymmetric non-atomic keys, one-way functions and inequalities. Moreover, some kind of support is provided for operators with algebraic properties, for example exponentiation.

HLPSL specifications are then translated into an intermediate language, IF (Intermediate Format), which is

1: $A \rightarrow S: \; S, A$
2: $S \rightarrow A: \; A, n_S$
3: $A \rightarrow S: \; S, \{n_S\}_{k_{shared}}$
4: $S \rightarrow A: \; A, OK$

Figure 2. Shared Key Authentication Protocol

used to carry out the analysis by means of a software tool written in Haskell.

The intruder is modelled implicitly and conforms to the Dolev-Yao model, with the additional ability (that has to be specified explicitly) for the intruder to assume the role of a legitimate protocol participant.

OFMC is based on two complementary techniques: the *lazy intruder* and the *lazy demand-driven search*. The former is the OFMC customisation of symbolic techniques, while the latter provides a finite representation of an infinite state space, because each portion of the state space is really computed only when it is being analysed. As a consequence, there is not any *a priori* limit set on the depth of the state space exploration, although such a limit is still needed to ensure the termination of the analysis on a protocol with no flaws.

Recent work on OFMC has been focused on the integration of reduction techniques based on partial-order reduction [16], and the eventual introduction of heuristics is foreseen to further improve the efficiency of the analysis. Moreover OFMC can deal with typed terms, in order to keep the state space smaller, even if it entails lacking the ability of detecting type-flaw attacks.

3 The 802.11 security algorithms

The 802.11i standard [17] enhances the security mechanisms described in [1]. In particular the security mechanisms of [1], now called *pre-RSNA algorithms*, although still supported for backward compatibility, have been deprecated and supplemented by new ones, the *RSNA algorithms*. In the remaining part of this paper two protocols of the 802.11 suite will be described and formally analysed, that is the *Shared Key Authentication protocol*, and the *Four-Way Handshake protocol*, belonging to the *pre-RSNA* and *RSNA* classes respectively.

3.1 Shared Key Authentication protocol

The authentication mechanisms defined in the 802.11 standard are used by access points (APs) to authenticate stations (STAs) before they are allowed to form an association. Obviously, no real authentication is provided with the Open System Authentication mechanism, since anyone who knows the SSID of an AP can gain access to it. In turn, discovering the SSID of an AP is really easy because each AP broadcasts a *beacon* management frame, containing its SSID as a clear text, at fixed time intervals.

On the other hand, Shared Key Authentication was designed to be secure, but a couple of important attacks were found and documented in the past. The first one can occur because the Shared Key Authentication mechanism does not provide mutual authentication. If a rogue AP is placed in a network, it can fraudulently complete fake authentication sequences with legitimate STAs.

The second flaw, described in [18] and [19], allows an attacker to authenticate itself after intercepting a single authentication sequence. This attack is due to the fixed length of the authentication frames and to a well known weakness in the WEP encryption mechanism [18, 20], which enables the creation of properly encrypted WEP messages without any knowledge of the encryption key. All the attacker needs is a plaintext-chipertext pair of suitable length, which can be obtained by eavesdropping a single legitimate authentication sequence. Since this attack depends on a weakness of the cryptosystem, the tools considered in this paper, which rely on the perfect encryption assumption, cannot discover it.

An abstract version of the Shared Key Authentication protocol is depicted in Fig. 2 by means of the commonly used “Alice & Bob” notation: two agents, A and S, already sharing the symmetric key k_{shared}, authenticate each other by agreeing on nonce n_S.

Informally, the protocol proceeds as follows:

A wants to authenticate with S;

S generates the nonce n_S and sends it to A;

A encrypts it with k_{shared} and sends it back to S;

S issues a successful acknowledge to A after having decrypted the message and having found n_S.

3.2 IEEE 802.11i RSNA and 4-Way Handshake

In order to overcome the deficiencies of the WEP encryption mechanism mentioned above and offer a more secure solution to wireless links, [17] specifies Robust Security Network Associations (RSNAs), which provide functions for both protecting data frames and performing authentication and key management based on the joint adoption of 802.11 and 802.1X [21]. However, to preserve backward compatibility, 802.11i does not forbid the implementation of WEP, even though its use is discouraged in favour of the newer methods.

The standard defines several ways for a local station to establish an RSNA, but in all of them the so-called *4-Way Handshake* key management protocol plays a central role because it is responsible for completing the 802.1X authentication process, confirming mutual possession of the same Pairwise Master Key (PMK) between two associated stations, allowing these stations to derive a fresh Pairwise Transient Key (PTK), and distributing the Group Temporal Key (GTK). The four messages used in the protocol are EAPOL-Key message fields, as specified by [21].

An abstract model of the protocol is shown in Fig. 3, where:

A and S represent the authenticator (an AP) and the supplicant (an STA), respectively, whose Medium Access Control (MAC) addresses are AA and SPA;

$$\begin{array}{lll}
1: & A \rightarrow S: & AA, n_A, sn, msg_1 \\
2: & S \rightarrow A: & SPA, n_S, sn, msg_2, \mathrm{H}(PTK, n_S, sn, msg_2) \\
3: & A \rightarrow S: & AA, n_A, sn+1, msg_3, \\
 & & \mathrm{H}(PTK, n_A, sn+1, msg_3) \\
4: & S \rightarrow A: & SPA, sn+1, msg_4, \mathrm{H}(PTK, sn+1, msg_4)
\end{array}$$

where $PTK = \mathrm{H}(PMK, AA, SPA, n_A, n_S)$

Figure 3. The 4-Way Handshake Protocol of 802.11

- n_A and n_S are two nonces, generated by the authenticator and the supplicant;
- sn is a serial number representing the Key Replay Counter field of the EAPOL-Key frame;
- the items msg_i represent portions of the messages, transported by the Key Data field of the EAPOL-Key frame, that have not been modelled explicitly;
- the abstract, one-way function $\mathrm{H}(\;)$ is used to model the computation of the 802.11 Message Integrity Code (MIC);
- the same function is also used to model the 802.11 function that derives PTK by combining PMK, the authenticator and supplicant MAC addresses, and the nonces just exchanged.

Informally, the protocol proceeds as follows:

- Firstly, the authenticator generates its own nonce, n_A, and sends it to the supplicant with protocol message 1, along with other information; albeit not captured by our model, this message also carries the identifier of the PMK to be used. It should be noted that, since PTK is still unknown to the authenticator, this message has an empty Key MIC field.
- Upon reception of this message, the supplicant checks the validity of the Key Replay Counter; if the check succeeds, the supplicant has now enough information to build PTK.
- The supplicant generates its own nonce n_S, assembles protocol message 2 and sends it to the authenticator; since PTK is now known to the supplicant, it can fill the Key MIC field of this message appropriately.
- When the authenticator receives protocol message 2, it checks that its Key Replay Counter matches the one sent in message 1, computes PTK and, with this information, verifies the MIC of the message.
- Then, the authenticator builds protocol message 3 and sends it to the supplicant. The Key Data field of this message encapsulates GTK, and has a valid Key MIC field.
- The supplicant now verifies that the MIC of the received message is correct, and that its Key Replay Counter and n_A are consistent with those received in message 1. Provided these checks succeed, the supplicant can start using PTK by configuring it into the 802.11 MAC.
- Finally, the supplicant sends protocol message 4 to the authenticator. When the authenticator receives this message, it verifies it has got a valid MIC and Key Replay Counter, and starts using PTK in its turn.

$$\begin{array}{lll}
1: & A \rightarrow I(S): & S, A \\
 & I(\Theta) \rightarrow S: & S, \Theta \\
2: & S \rightarrow I(\Theta): & \Theta, n_R \\
 & I(S) \rightarrow A: & A, n_R \\
3: & A \rightarrow I(S): & S, \{n_R\}_{k_{shared}} \\
 & I(\Theta) \rightarrow S: & S, \{n_R\}_{k_{shared}} \\
4: & S \rightarrow I(\Theta): & \Theta, OK
\end{array}$$

Figure 4. The First Attack on the 802.11 Shared Key Authentication Protocol

Due to its importance, the 4-Way Handshake protocol was already analysed in [22], using a finite state verification tool, and a Denial of Service (DoS) attack was found. However, the tools being considered in this paper are mainly suited to find safety errors; thus, liveness errors like denials of service are not caught.

4 Analysis of the 802.11 security mechanisms

4.1 Formal analysis of the Shared Key Authentication protocol

In this section we describe two authentication flaws (no further safety flaws are known for this protocol) that all tools discovered automatically. In particular, the protocol introduced in Fig. 2 was checked and Fig. 4 shows the first attack we have found. In the picture, $I(x)$ means that the attacker behaves as agent x. The authentication attack (introduced in Sect. 3.1) proceeds as follows:

- When A starts a session of the protocol and sends its first message, the attacker intercepts and relays it to the intended recipient S after replacing the true initiator's identifier A with a fake identifier Θ.
- When S replies, the attacker again intercepts and relays the answer to A after restoring the mobile station identifier to the value A expects.
- At this point, the attacker lets the protocol proceed as expected up to the end.

Even from a rough analysis of the exchanged messages, it can be noted that something in the protocol went wrong, because the fourth message of the session (indicating the successful completion of the authentication sequence) carries the fake identifier Θ, chosen by the attacker, instead of the intended identifier A.

1: $A \to I(S) : S, A$
2: $I(S) \to A : A, \gamma$
3: $A \to I(S) : S, \{\gamma\}_{k_{shared}}$
4: $I(S) \to A : A, OK$

Figure 5. The Second Attack on the 802.11 Shared Key Authentication Protocol

More precisely, it is possible to gain a better understanding of the flaw in the authentication sequence by comparing the very different points of view the honest mobile station A and the access point S have about the outcome and the meaning of the protocol session just shown:

> Station A tries to authenticate itself with S but it does not receive the final acknowledgement (and will never receive it). Hence, the authentication attempt is unsuccessful and A will have to try it again, probably after a timeout expires.
>
> S authenticates a mobile station Θ successfully, even if such a station does not know the secret key shared between A and S.

The second protocol weakness, which has been found, is more subtle, and is shown in Fig. 5: in this case the attacker, masquerading as an access point, can engage A in an authentication session. As a consequence, the attacker can use A as an "oracle" to encrypt a datum of its own choice (denoted with γ in the figure and passed to A in place of the challenge text) with the secret key it does not know. Then, the attacker could use this information in further communication steps, for example after obtaining a successful authentication with the honest access point by exploiting the first attack.

On the other hand, A is convinced it carried out a run of the authentication protocol successfully with the honest base station S, but this is totally useless because S could not even be aware of the presence of A in the network at this point.

As we mentioned before, the two attacks were found using a simplified specification of the 802.11 Shared Key Authentication protocol. Now, let us discuss the feasibility of carrying them out in practice, taking all the low-level aspects into account.

One first question is whether an attacker can really put these attacks into practice. First of all it is worth noting that all the necessary hardware to monitor and inject 802.11 traffic is readily available to consumers in the form of conventional wireless network cards and the necessary software is available on the internet. The only point that should be considered carefully is that the attacker could have trouble in modifying an intercepted message transmitted over the wireless medium. However, this is possible in scenarios where the access point S is outside the radio coverage of the mobile station A and vice-versa, while the attacker can reach both A and S. In this way, the attacker can not only intercept messages transmitted between A and S, but also modify them before relaying them to the other party for processing.

Scenario	**Tool**			
	S^{3}A	**OFMC**	**STA**	**Casper**
1A 1S	0.09 s	0.07 s	0.53 s	25, 10 s
2A 1S	21.25 s	0.15 s	23.26 s	161.34 s
2A 2S	—	1.92 s	—	626.78 s
3A 2S		10.51 s		3797.18 s
3A 3S		—		—

Table 2. Execution Times on the 4-Way Handshake Protocol

A second point to be investigated is whether the attacks we found for the simplified version hold for the real protocol, too. To make this point clear, we wrote a detailed model that is closer to the standard protocol specification [1]. In particular, it also includes message sequence numbers, protocol selection tags and additional status codes neglected in the simplified model.

By analysing the detailed model we discovered the same attacks found for the simplified model. In particular, the first attack, depicted in Fig. 4, maintains the same structure except for the format of the messages which includes the fields neglected in the simplified specification. The same conclusion applies to the second attack shown in Fig. 5 for the high-level version of the protocol, but, in this case, it must be pointed out that the message the attacker obtains by using agent A as an oracle (γ $_{k_{shared}}$) contains other fields besides γ: in fact, the corresponding message of the detailed version is $SK, 3, \gamma$ $_{k_{shared}}$, where SK is a tag indicating that the message belongs to a Shared Key Authentication handshake, while the second field taking value 3 tells that this is the third message of the authentication procedure.

Another question to be answered about the impact of low-level details on the attacks we discovered concerns their compatibility with the frame format of the 802.11 protocol. In particular, this must be checked for the second attack, where authentication messages are replayed as data messages. As shown in Fig. 6 the attack can really be carried out in practice: in fact, the frame body of the management message sent from the STA to the AP can easily be extracted from the message and inserted into a data frame having a body length of 144 octets. The only restriction is the one already found with the detailed model, i.e. that the plain text corresponding to the encrypted text contained in the new message must have a constant prefix of 8 octets, representing the first two fields of the message.

4.2 Formal analysis of the 4-Way Handshake protocol

The focus of the analysis for this protocol has been on the agreement on the value of the secret PTK, reached by the right protocol parties after the protocol has run. In

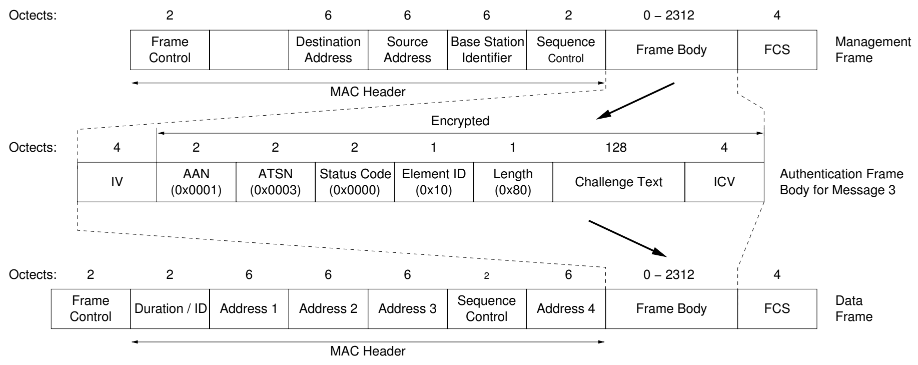

Figure 6. Feasibility of the Type-Flaw Attack

fact, this is one of the crucial protocol goals as mentioned in Sect. 3.2.

Although all considered tools assessed that the protocol actually satisfies this property, different amounts of user effort and some workarounds were needed to reach this result: in this case more attention must be paid when the reliability of the results is checked, moreover these workarounds are easy to be set up only when the flaw to be catched is known in advance. This aspect was made even harder for some tools due to the higher complexity of the messages of the 4-Way Handshake protocol. In particular:

STA has some problems handling nested hash functions, hence they have been replaced with a public-key encryption using a fake key whose private part was guaranteed to be unknown to anyone.

OFMC and Casper do not support the presence of non-atomic terms in the specification of the properties to be verified. In the first case, this capability was not foreseen by the specification language syntax; on the other hand, albeit the Casper syntax allows it, the compiler failed. Hence, it was not possible to specify an agreement on the value of the secret PTK directly and we resorted to specify a weaker property, namely an agreement on the value of PMK.

Instead S^{3}A successfully checked the specification against the requirement without the need of shortcuts or workarounds. This means that S^{3}A is able to do a finer-grained analysis than STA and OFMC, at the greater expense of computational resources. In fact, as shown in Table 2, Casper and OFMC carried out the analysis up to a scenario with three concurrent instances of the authenticator and supplicant roles (3A3S), whereas S^{3}A and STA exceeded the upper limit set on resource consumption (2 hours of CPU time on a Pentium IV class machine) for a scenario with two concurrent instances of the authenticator and supplicant roles (2A2S).

OFMC appears to be the best tool among those considered because its resource requirements, due to the effectiveness of the lazy evaluation techniques it is based on, are quite small.

Also Casper behaves very well and the reason may lie in the upper bound enforced by Casper on the length of the messages generated by the intruder; this approach shrinks the portion of the state space the tool actually explores significantly. This behaviour affects the accuracy of the analysis undoubtedly, but can also make the analysis time considerably shorter.

On the other hand, the limited performance of S^{3}A can be partially justified if we observe that this tool carries out a testing equivalence check that is more powerful, but also more expensive, than the checks based on reachability analysis, as already stated in Section 2.2.

5 Concluding Remarks

The ICT (r)evolution is deeply affecting several aspects of the traditional industrial scenarios. On the one hand, the widespread use of the internet and the availability of large public communication networks enable the introduction of several appealing features in the area of the distributed control systems, where functionalities such as the remote supervision, device monitoring, diagnostics and maintenance are ever more demanded in a number of hi-tech applications.

Unfortunately openness, reachability and accessibility also imply some significant disadvantages when the security of the overall system is considered, because advanced industrial systems are exposed to the attacks of hackers and malicious users as conventional networks are.

Generally speaking, the security of a system can and must be improved significantly by means of a careful design carried out by taking into account also the security requirements besides the other needs. In particular, it is important to analyse and check for correctness at least those parts of the system affecting the most critical security-related aspects. This applies, in particular, to the design and development of cryptographic protocols that can be considered as one of the main building blocks to be used when security services must be provided. In the past, formal specification and analysis techniques, because of their abstract and mathematics-based characteristics, were studied and adopted mainly by researchers and computer scientists to deal with simple "toy" examples. More recently, however, some interesting automatic/semi-automatic software tools have begun to appear, that are able to deal with case studies of reasonable size and complexity. Even though these tools have not been engineered and distributed commercially yet, nevertheless they can be considered as a new first generation of computer-aided supports to the development of cryptographic protocols.

This paper has presented the results of the investigation of a selection of such tools, specifically designed to analyse cryptographic protocols. The presented research has been focused on available, fully automatic tools. Through a case study, it has been shown how the selected tools can be applied to investigate the security of typical authentication mechanisms, such as the ones included in the 802.11 standards, which are widely adopted in industrial environments. All the selected tools have been able to find out the known logical weaknesses of the Shared Key Authentication protocol, thus indicating their maturity and usefulness in analysing real protocols. The differences found among the different tools mainly regard their performance, e.g. are related to the time needed to perform the analysis or to the number of concurrent sessions that can be analysed, and their flexibility in dealing with the various protocol features or security properties.

A final remark is that the full automation of the analysis process can be a key point for a more widespread adoption of solutions based on formal techniques, because it reduces the required level of expertise dramatically.

References

[1] IEEE Computer Society, New York, *ANSI/IEEE Std 802.11-1999 (Reaff. 2003) Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless Lan Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.

[2] J. Clark and J. Jacob, “A Survey of Authentication Protocol Literature: Version 1.0”, Available online, at `http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997.

[3] D. Dolev and A. Yao, “On the security of public key protocols”, *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[4] L. Durante, R. Sisto, and A. Valenzano, “Automatic Testing Equivalence Verification of Spi Calculus Specifications”, *ACM Trans. Softw. Eng. Meth.*, vol. 12, no. 2, pp. 222–284, 2003, DOI 10.1145/941566.941570.

[5] G. Lowe, “Towards a completeness result for model checking security protocols”, *J. Comput. Sec.*, vol. 7, no. 2–3, pp. 89–146, 1999.

[6] R. Needham and M. Schroeder, “Using Encryption for Authentication in Large Networks of Computers”, *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[7] A. W. Roscoe, *A Classical Mind, Essays in Honour of C. A. R. Hoare*, chapter Model-checking CSP, International Series in Computer Science. Prentice Hall, Hertfordshire, 1994.

[8] C. A. R. Hoare, *Communicating Sequential Processes*, International Series in Computer Science. Prentice Hall, Englewood Cliffs, 1985.

[9] G. Lowe, “Casper: a compiler for the analysis of security protocols”, in *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW 1997)*, 1997, pp. 18–30, Washington. IEEE Computer Society Press, DOI 10.1109/CSFW.1997.596779.

[10] R. De Nicola and M. C. B. Hennessy, “Testing equivalence for processes”, *Theor. Comput. Sci.*, vol. 34, no. 1-2, pp. 84–133, 1984.

[11] M. Abadi and A. D. Gordon, “A Calculus for Cryptographic Protocols: The Spi Calculus”, *Inf. Comput.*, vol. 148, no. 1, pp. 1–70, 1999, DOI 10.1006/inco.1998.2740.

[12] R. Milner, J. Parrow, and D. Walker, “A Calculus of mobile processes, parts I and II”, *Inf. Comput.*, vol. 100, no. 1, pp. 1–77, 1992, DOI 10.1016/0890-5401(92)90008-4.

[13] M. Boreale and M. G. Buscemi, “Experimenting with STA, a Tool for Automatic Analysis of Security Protocols”, in *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, 2002, pp. 281–285, New York. ACM Press.

[14] M. Boreale and M. G. Buscemi, “A Framework for the Analysis of Security Protocols”, in *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR 2002)*, volume 2421 of *Lecture Notes in Computer Science*, 2002, pp. 483–498, Berlin. Springer-Verlag.

[15] D. Basin, S. Mödersheim, and L. Viganò, “OFMC: A Symbolic Model Checker for Security Protocols”, *Int. J. Inf. Secur.*, vol. 4, no. 3, pp. 181–208, 2005, Special issue on ESORICS 2003.

[16] D. Basin, S. Mödersheim, and L. Viganò, “Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols”, in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, 2003, pp. 335–344, New York. ACM Press.

[17] IEEE Computer Society, New York, *IEEE Std 802.11i-2004 Medium Access Control (MAC) Security Enhancements, Amendment 6 to IEEE Standard for Information technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2004.

[18] N. Borisov, I. Goldberg, and D. Wagner, “Intercepting Mobile Communications: The Insecurity of 802.11”, in *Proceedings of the International Conference on Mobile Computing and Networks*, 2001, pp. 180–189.

[19] W. A. Arbaugh, N. Shankar, and Y. C. J. Wan, “Your 802.11 Wireless Network has No Clothes”, in *Proceedings of the First International Conference Wireless LANs and Home Networks*, 2001, pp. 131–144.

[20] J. Walker, “Unsafe at any key size: An analysis of the WEP encapsulation”, 2000, IEEE 802.11 Task Group E IEEE 802.11/00-362.

[21] IEEE Computer Society, New York, *IEEE Std 802.1X-2004 Local and Metropolitan Area Networks - Port-Based Network Access Control*, 2004.

[22] C. He and J. C. Mitchell, “Analysis of the 802.11i 4-Way Handshake”, in *Proceedings of the 2004 ACM Workshop on Wireless Security (WiSe’04)*, 2004, pp. 43–50, New York. ACM Press.