A formal methodology for integral security design and verification of network protocols

Jesus Diaz, David Arroyo, Francisco B. Rodriguez

September 7, 2012

Abstract

We propose a methodology for verifying security properties of network protocols at design level. It can be separated in two main parts: context and requirements analysis and informal verification; and formal representation and procedural verification. It is an iterative process where the early steps are simpler than the last ones. Therefore, the effort required for detecting flaws is proportional to the complexity of the associated attack. Thus, we avoid wasting valuable resources for simple flaws that can be detected early in the verification process. In order to illustrate the advantages provided by our methodology, we also analyze three real protocols.

1 Introduction

With this work we would like to promote dissemination of methodologies for the verification of security properties of interactive protocols. It covers the whole protocol design process, from the specification of the requirements and the protocol environment, to the protocol definition and its validation. Such a methodology will help avoiding errors due to wrong or no longer valid assumptions on the context in which the protocol is executed, while also preventing design flaws that can lead to the non-fulfillment of the security requirements. Indeed, there already exist plenty of tools for the formal analysis of those security requirements [8, 4]. However, even though there exist automatic tools for formal security verification, dedicating some time previous to this formal analysis to the task of defining a security context may help to detect some initial (but important) flaws. This way, we also prevent an unnecessary waste of time and improve the understanding of the protocol. Like we will see, an active application of a methodology covering the whole design process will help reducing the impact of illegitimate actions over the resulting network protocols.

The paper is structured as follows. We start in Sec. 2 with a brief introduction to existing frameworks and procedures for verification of security properties, along with a discussion on the advantages of applying the verification procedures at the different stages of the software life cycle. In Sec. 3

we introduce our methodology. In Sec. 4 we apply the different phases of our methodology to three different protocols (MANA III, WEP-SKA and CHAT-SRP), and we conclude this work in Sec. 5, with a global perspective of the benefits provided by our methodology.

2 Related work and discussion

Security flaws in information systems are receiving increasing attention, as the problems and losses they can cause grow in severity [29, 30, 33]. As a result, organizations are paying more attention to the design of secure systems. In that matter, several frameworks have been developed during the last years, intended to help in the task of creating secure software by delimiting where can security flaws be originated [31, 22, 25]. There are also proposals and standards defining different security levels that may be required depending on the context and managed information, and even the methods and tools used for verification [1, 27]. While [31, 22, 1] are aimed for every software system and all the components within it, the proposal in [27] is centered in evaluating security in cryptographic protocols. Certainly, cryptographic protocols are quite important for the evaluation of security properties, since they are applied when something is required to be protected. Moreover, protocols can be seen as components that need special consideration when evaluating security because they are subject to specific risks that can hardly take place in other components (e.g., replay, spoofing or Man In The Middle attacks). In the mentioned work, several levels of security are defined, depending on the level of formalization applied in the different elements of the methodology (protocol specification, adversarial model, security properties and self-assessment evidence), and the type of tools used in the process. Indeed, we shall refer to them [27, Sec. 2.1] for a good review of the different type of formal methods and tools that can be used for cryptographic protocols analysis.

Besides the classification of verification tools in terms of the theory they are based upon, there is yet another further division: the point in the software development life cycle where they are applied. Precisely, they can be applied to check the design obtained after the design phase; or to check the code created during the implementation phase. Examples of tools for verification at the design phase are ProVerif, CryptoVerif, CertiCrypt, Cryptyc, AVISPA and Isabelle [8, 9, 5, 4, 12, 28]. On the other hand, examples of code verifiers are [7, 11, 21]. As usual, there are supporters for each of the different alternatives. Our point of view is that neither of them should be neglected, for the following reasons. Imagine that we just choose to apply a verification tool directly to the obtained system implementation, and we indeed find security flaws. Now suppose that, while trying to fix them, we realize that one (or several) of them is not just a coding flaw, but a design flaw. That means that we must go back to the design phase and fix the flaw at that stage. Moreover, if the flaw is serious enough, the existing implementation may need extensive changes, which will incur in unacceptable costs and delays. Yet another disadvantage is that code verification obviously relies on the existence of a tool for verifying the specific programming language that has been used. Although these tools are gaining popularity, the huge number of existing programming languages, along with the complexity of creating such a tool, suggests that, in the real world, we should not trust in having always a code verification tool suitable to our needs. On the other hand, deciding just to apply a verification tool to test if our design is robust does not guarantee that the implementation of that design will also be secure even though the design seems to be so. However it is technology independent. Thus, we argue that verification of the security properties of both the design and its implementation should be applied when possible.

We propose here a simple, but yet powerful, methodology for the verification of the design of communication protocols. Thus, we keep in mind the specific risks that we can encounter when desiging them, but set aside the more general software flaws that, although can ultimately affect the security properties of the protocols, should be dealt with by the tools applied for code verification. As a methodology, its main advantage is that it is directly devised to help detecting simple flaws early in the design process, leaving only the complex, time-consuming ones for the last steps. Thus, it avoids wasting too much time in flaws that could have been easily detected. It also allows reaching the highest levels of assurance of other frameworks, like [27, 1], depending on the tools used for procedural verification.

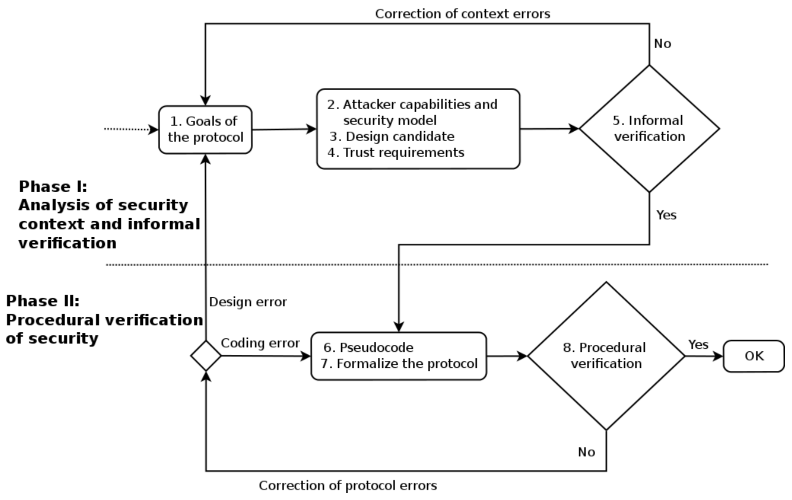
3 The methodology

Our methodology consists of two main parts, like shown in Fig. 1. The first part is concerned with specifying the security requirements and considering if they are suitable in a given context, and performing a first informal verification; while the second part is centered in the formalization and formal verification of the protocol itself.

3.1 Analysis of security context and informal verification

This is the first part of the methodology and is mainly intended to avoid incongruences related to security requirements unachievable or inappropriate for some reason, and to improve our understanding of the protocol. It also allows us to perform several informal checks of our first design candidates. This part is depicted in the first phase of Fig. 1. Namely, here we face the following matters:

Step 1. Goals of the protocol. This step might seem too obvious to be needed, but specifying the goals of the protocol will help to improve the understanding of the protocol, and will also come in handy for the third and fourth steps. As output here, we will get a rather informal specification of what we intend to achieve with our protocol, something like *"Complete certainty that the person being registered in our website is who he/she says he/she is"*.



Correction of context errors
No
1. Goals of the protocol
2. Attacker capabilities and security model
3. Design candidate
4. Trust requirements
5. Informal verification
Phase I: Analysis of security context and informal verification
Yes
Phase II: Procedural verification of security
Design error
Coding error
6. Pseudocode
7. Formalize the protocol
8. Procedural verification
Yes
OK
No
Correction of protocol errors

Figure 1: Steps of the proposed methodology. From the initial design of the protocol, the first phase consists of performing an analysis of its goals, the security model that best fits our needs a design candidate, and the trust requirements we expect to achieve. During the second phase, after giving informal and formal definitions of the protocol, we apply the chosen tools or models for the verification of the security requirements established before. Note that the methodology is structured as a loop, which is repeated as long as new context, protocol errors are detected.

Step 2. Attacker capabilities and security model. This step is essential, because depending on the capabilities the attacker has, some security requirements might or might not be achievable. In this sense, there are two main attacker models: the Dolev-Yao attacker [17] and the computational attacker [6]. The Dolev-Yao attacker model assumes perfect cryptographic primitives but gives otherwise full control to the attacker over the communications network. The computational attacker is somehow more realistic in that it is assumed to be a Probabilistic Polynomial-time Turing Machine. But, as a matter of fact, we are also worried about other attacker capabilities, like the capability to access public records containing personal information assumed to be secret in the protocol like Facebook pages, lists of personal passwords published by hackers or leaked, etc.). Indeed, this kind of information is usually assumed to be secret in the typical attacker models, and it can drastically change the provided security guarantees. On the other hand, we might find ourselves in a situation where all the capabilities of the typical Dolev-Yao or computational attacker do not hold. So, in short, here we have to ask ourselves whether the assumptions made by standard attacker models are close to our reality, or if we need to adjust them in some way. May we ignore this step, any later proof might be wrong because our initial assumptions were not close to our reality [3]. As a result, with this step we will get a specification of the attacker model, e.g. Dolev-Yao or computational, along with a list of specific capabilities added or subtracted to it, like *"Knows the home address of everyone that may be involved in the protocol"* or *"Cannot eavesdrop wired communications"*. Henceforth, we will refer to this list as the *"+/- capabilities"* list.

Step 3. Design candidate. Once we have made up our mind as to which are the goals of our protocol and the attacker capabilities and security model we assume, we can adventure ourselves to cast a design candidate. For communications protocols, a sequence diagram is most commonly used, since it captures both the order of the messages composing the protocol and their contents. Therefore, as output of this step we will obtain a sequence diagram depicting the protocol.

Step 4. Trust requirements. This step of the methodology is the first part of an informal verification procedure. Besides helping us in checking the security requirements, it is also intended to prevent us from wasting our efforts needlessly. Before undertaking the formal verification task, which may be time consuming, it is worth making a simpler manual verification, in order to detect obvious mistakes. For that purpose, we will follow the design candidate from the beginning, asigning trust properties. Note that we use the term trust property rather than security property because at this point, this properties are something we expect the elements to keep, not something we state that they have. The procedure is as follows: for each step of the sequence diagram, we will note down the trust properties we expect from every of its elements to keep. If, within a specific step, we apply some function to merge several elements or disassemble them, we may have to require additional trust properties to the result (e.g., a message containing the a priori unauthenticated elements x_1, x_2 along with $MAC_K(x_1, x_2)$ can be considered authenticated, by the holder of a key K, if the MAC verification suceeds, provided that the key K is trusted)[1]. Since the properties we require from a specific element may change through the protocol, we will revise them for every step, even if the element has already been processed previously. This process is summarized in Algorithm 1. Also, the requirements that we may need to apply to the different elements are[2]:

None. When a field does not require any specific trust property.

Authenticity. When the authenticity of the field must be guaranteed.

Confidentiality. When the specific field must be kept secret.

[1]Typically, this kind of rules are specified formally inside formal verification tools (e.g., Cryptyc [4] is based in type deduction rules). However, since this step is intended to be a preliminary informal approach, common sense and experience are enough.

[2]Although we propose this five requirements, more could be defined if necessary.

Integrity. When the integrity of the field must be preserved. Detecting any undesired modifications.

Uniqueness. When an element cannot be repeated among the same or different protocol runs. This property is commonly used to guarantee freshness and avoid replay attacks.

```
Data: The sequence diagram of the protocol.
Result: A set of trust requirements for each protocol step.
for $s$ = first step; until $s$ = last step do
    for $e$ = first until $e$ = last element of step $s$ do
        Set reqs[s][e];
    end
    reqs[s] += Additional trust requirements for step s;
end
```

Algorithm 1: Algorithm for assigning requirements to protocol elements and messages. With elements, we refer to any component calculated or sent within each specific step.

Therefore, as output of this step we get a list of trust requirements which, optionally, can be depicted jointly with the sequence diagram to create a requirement-tagged sequence diagram of our protocol design candidate.

Besides, we also have to decide in this step which verification tool or process we will apply in the second phase. Now that we know our specific trust requirements, we can choose an appropriated tool, since some tools may not be applicable (depending on the security properties we want to verify).

Step 5. Informal verification. Now we have to take the +/- capabilities list and the list of trust requirements (or the requirement tagged sequence diagram, whichever representation we are using). If any of the trust requirements enters in conflict with any of the capabilities we granted to the attacker, then we have a security failure. If not, we have informally verified the design candidate and we can continue. However, we must keep in mind that this does not guarantee that the design will also pass the next phase of the methodology.

After applying our methodology up to this point for verifying a protocol, and assuming we passed the step 5, we would have reached the assurance level PAL1 defined in [27].

3.2 Procedural verification of security

This part of the methodology is devoted to the procedural verification of the security requirements established before. It is depicted in the second phase of Fig. 1. With procedural verification we refer to the fact that widely approved theories, methods or procedures should be applied here. Again, we can apply either the formal or the computational model. The main advantage of the formal methods is that there are plenty of tools that automatically analyze a protocol formalization. On the other hand, the computational model verification takes into account that the cryptographic primitives may not be perfect. However, in the last years, there has been a lot of work to prove that formal security implies computational security [2, 24], given that the cryptographic primitives meet some properties. We do not enter in this matter here, and just assume that a "procedural" verification model has been adopted and is going to be applied. Indeed, the aim of our methodology is to formally tackle the verification of the security properties of communication protocols. Nevertheless, we have found that the application of automatic tools (like ProVerif [8], Cryptyc [4], etc.) allows to detect many flaws with little effort, so it may be a good choice to first apply formal methods and, if required, after that, use the computational model for more concrete evaluation.

Therefore, the steps we have included in this phase are as follows:

Step 6. Protocol pseudocode In the same way that writing pseudocode is useful before coding a program, writing an informal narration of a protocol in the shape of pseudocode, helps to reach a higher concretion level before properly formalizing it. As output of this step we get a written representation of the sequence diagram, with one process for each principal, which depicts the internal computations performed by each of them in order to generate the messages' components, along with the messages sent to the other principals.

Step 7. Formalization of the protocol From the protocol pseudocode, it is typically easy to produce a formalization in the language or definition model required by the tools we are going to use to verify the protocol (the ones we decided to use in step 4). The result of this step must be a meticulous representation of how the protocol will be, once implemented.

Step 8. Procedural verification The formalization obtained in the previous step will then be used as input for the chosen procedural verification model or tool. If the tools or procedures followed in this step "output" that all requirements are fulfilled, we can conclude. If not, we have to go back to the first step and correct protocol errors in our design, checking also for coding errors in the formalization.

As an end note, we must point out that the methodology does not give as output an explicit measure of security of the analysed protocol, other than checking whether the specified security requirements are held. That will depend on the tools or the models used to verify the protocol. In any case, at least, we will obtain an answer to whether or not the protocol meets the requirements specified in step 4 of our methodology. Again referencing [27], any protocol successfully verified using our methodology would reach an assurance level equivalent to PAL2 or PAL3, depending on whether the used tool provides bounded or unbounded verification (or even the - yet under consideration in [27] - PAL4 level, would we have used a computational model verification tool).

To summarize, the inputs and outputs of each of the different steps of our methodology are shown in Tables 2 and 3, corresponding to the first and second phases of the methodology. The acronyms used in the tables for the different outputs are explained in Table 1.

O1:	Informal list of goals.
O2:	Security model and +/- capabilities list.
O3:	Sequence diagram.
O4:	List of trust requirements / Requirement-tagged sequence diagram.
O5:	Pseudocode.
O6:	Formalization.

Table 1: Definition of the different outputs of the methodology.

	Goals	Attacker model and capabilities	Design candidate	Trust requirements	Informal verification
Input:	None	None	O1	O3	O2 and O4
Output:	O1	O2	O3	O4	Yes/No

Table 2: Inputs and outputs of each step of the first phase of the methodology.

	Pseudocode	Formalization	Formal verification
Input:	O4	O2 and O5	O6
Output:	O5	O6	Yes/No

Table 3: Inputs and outputs of each step of the second phase of the methodology.

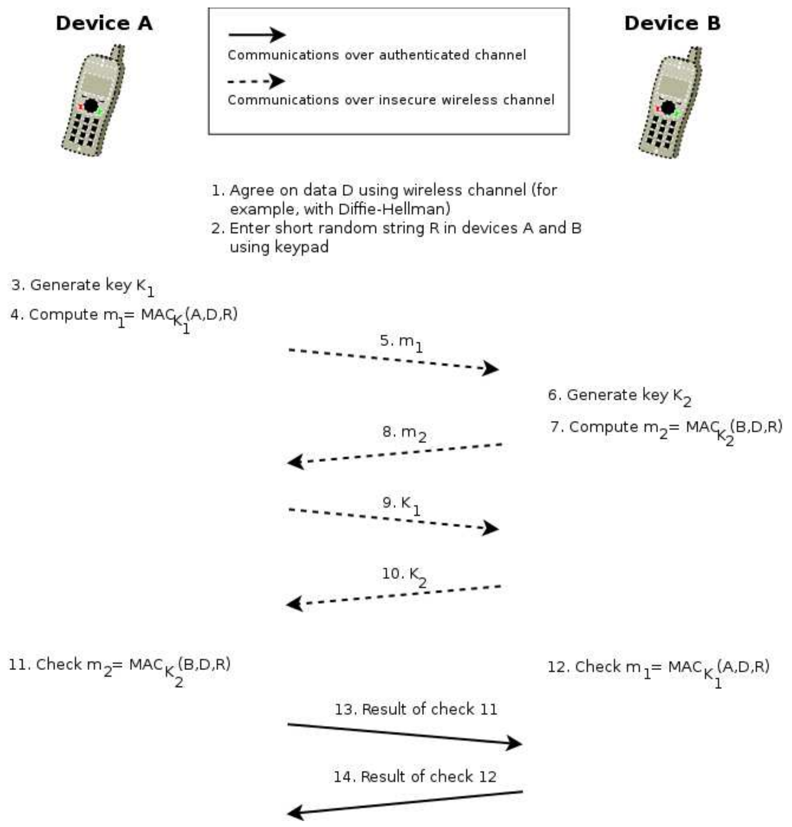
4 Case studies of real security protocols

In this section we provide three case studies by applying our methodology to real security protocols. For the first case, a problem is located in the first phase of the methodology, while in the second case, a problem is detected with a formal analysis in the second phase.[3] In the other hand, in the third case, we go through the whole methodology to verify one more protocol [16, 15] which successfully passes our tests.

[3]It is not our purpose to present new flaws here. We rather intend to show how to avoid security design flaws using our methodology.

4.1 Case study I: context verification failure

Here we apply the first part of the methodology to the MANA III (MANual Authentication) authentication protocol [20]. MANA III is intended for wireless networks and bases its security in the requirement of manually introducing a short bitstring (R in Fig. 2) via keypads, previous to the execution, which is kept secret. Since this short secret bitstring is used in the calculation of MACs, an attacker will not be able to create fake MACs for authentication in real time. The MANA III protocol is informally depicted in Fig. 2.



Device A
Device B
Communications over authenticated channel
Communications over insecure wireless channel
1. Agree on data D using wireless channel (for example, with Diffie-Hellman)
2. Enter short random string R in devices A and B using keypad
3. Generate key K_1
4. Compute m_1= MAC_{K_1}(A,D,R)
5. m_1
6. Generate key K_2
7. Compute m_2= MAC_{K_2}(B,D,R)
8. m_2
9. K_1
10. K_2
11. Check m_2= MAC_{K_2}(B,D,R)
12. Check m_1= MAC_{K_1}(A,D,R)
13. Result of check 11
14. Result of check 12

Figure 2: Message sequence of the MANA III protocol [20]. (1) Devices A and B agree on some data D (e.g., using Diffie-Hellman). (2) A secretly shared short random bitstring known in both devices with a keypad. (3-8) Both devices independently generate a key, compute a MAC of their identities, D and R and send the result to each other. (9-10) Both devices exchange the keys used in the previous MACs. (11-14) Both devices check if the received MACs are correct, and inform the other device of the obtained result. Messages 5,8,9 and 10 are sent over insecure channels, while messages 13 and 14 are sent over an authenticated channel.

In [32] the authors suggest that this authentication method is no longer secure due to the proliferation of CCTV cameras. Indeed, such cameras can be used to observe the short bitstring introduced via the keypad. If the secret short random bitstring becomes known to the attacker, he could use it to mount a Man In The Middle attack [32]. Let us now see how this fact could have been detected with the first part of our methodology.

Step 1. Goals of the protocol. We want to achieve a final state in wich Device A and Device B are certain that they are communicating with each other. That is directly translated to authenticity. Thus, our specific goals are:

1. Device A is successfully authenticated against Device B.
2. Device B is successfully authenticated against Device A.

Step 2. Attacker capabilities and security model. Here we have to decide an attacker model, let us assume that we adopt the Dolev-Yao attacker model (nevertheless, this decision does not affect what follows). As we said in Sec. 3.1 we have to analyze our specific context to see if our attacker would have some extra capabilities (or if we have to take out some capability from him). In the MANA III protocol, and given the current state of development of CCTV cameras, like it is said in [32], it is not safe to assume that, in any situation, a password introduced via a keypad will always remain secret. One could also make a quick search on the Internet and find several microcameras of roughly 15x5 milimiters. With cameras that small, an attacker could manage to record almost anything without the victim even noticing. Therefore, the more realistic decision of giving the attacker the capability to observe keyed passwords seems to be justified[4]. Besides, we will trust that the assumption made by the authors of the protocol stating that the short random string R prevents attackers from finding, in real time, collisions for the calculated MACs. Summarizing, we use the Dolev-Yao model, and our +/- capability list will be as shown in Listing 1

```
+  May observe keystrokes.
−  Cannot find MAC collisions in real-time without R.
```

Listing 1: +/- capabilities list for the Dolev-Yao attacker for the MANA III protocol.

Step 3. Design candidate. From the goals we defined in the first step, we come up with a design candidate in the shape of a sequence diagram. Suppose that we produce the diagram in Fig. 2[5].

[4]However, this may be a too restrictive scenario for many situations, but for illustration purposes, we will assume that we intend to use MANA-III in a security critical context.

[5]Obviously, in a real scenario, we will produce it at this step, and not before. We just showed the diagram in advance for introducing the case study.

Step 4. Trust requirements. With the sequence diagram in front of us, we apply Algorithm 1. As a result, we obtain the list of trust requirements shown in Listing 2 (we omit the complete process for brevity).

```
Initialization. A and B are publicly known values.
Step 1. D : Authenticity
Step 2. R : Confidentiality
Step 3. $K_1$ : Confidentiality
Step 4. $m_1 = MAC_{K_1}(A, D, R)$ : Authenticity
        ( $K_1$ is trusted by A since she has
          created it. )
Step 5. $m_1$ : None
        ( At this point, no one knows $K_1$ and $m_1$
          is sent over the public channel, so we
          cannot place any trust in this element. )
Step 6. $K_2$ : Confidentiality
Step 7. $m_2 = MAC_{K_2}(B, D, R)$ : Authenticity
        ( $K_2$ is trusted by B since she has
          created it. )
Step 8. $m_1$ : None
        ( At this point, no one knows $K_2$ and $m_2$
          is sent over the public channel, so we
          cannot place any trust in this element. )
Step 9. $K_1$ : None
        ( The used cannel is insecure, anyone can
          send this. )
Step 10. $K_2$ : None
        ( The used cannel is insecure, anyone can
          send this. )
Step 11. Nothing to do here.
Step 12. Nothing to do here.
Step 13. $check_{m_2}$ : Authenticity.
Step 14. $check_{m_1}$ : Authenticity.
```

Listing 2: List of trust requirements for MANA III.

Step 5. Informal verification. Going through the list of trust requirements shown in Listing 2 we soon find an incompatibility with the attacker capabilities: in the second step, we see that we require R to be secret (confidentiality). Still, we gave the attacker the capability to observe keystrokes, and since R is entered in Devices A and B via a keypad, we cannot consider it secret.

We should see now the importance of this part of the methodology, and why does it have to be applied with care. Although the standard attacker models provide a powerful framework to start with, there are cases where these models do not cover all the possibilities that the attacker may have available. Therefore, in order to detect possible flaws on the context of the protocol, we have to carefully establish the protocol goals, analyze the attacker capabilities given the current technologies, give a formal statement of the required security properties, and check if they hold given the previous facts. Once we successfully complete this phase, we can proceed with the second phase of the methodology, with a higher degree of certainty that we are correctly grounded the context of our protocol.

4.2 Case study II: procedural security verification failure

We dedicate this subsection to the procedural analysis of the Shared Key Authentication of the WEP standard (WEP-SKA from now on), as defined in [23]. Let us then assume that the protocol successfully passes the first phase of our methodology. In this case, we apply the automatic formal verifier ProVerif [8].

The WEP-SKA protocol is one of the two authentication methods supported by the WEP standard. A normal execution consists of four messages, between two stations. We will call them the Wireless Device (WD), which wants to be authenticated, and the Access Point (AP) which attends the WD's request. A typical protocol run is depicted in Fig. 3.

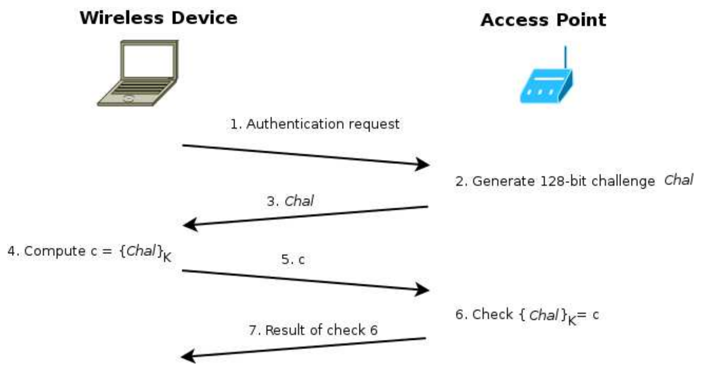

Wireless Device
Access Point
1. Authentication request
2. Generate 128-bit challenge *Chal*
3. *Chal*
4. Compute c = {*Chal*}$_K$
5. c
6. Check {*Chal*}$_K$= c
7. Result of check 6

Figure 3: Message sequence of the WEP-SKA protocol [23]. (1) The WD requests to be authenticated using WEP-SKA. (2-3) The AP generates a challenge, and sends it, in plaintext, to WD. (4-5) The WD encrypts the challenge with a preshared key, and sends the result to the AP. (6-7) The checks the received encrypted challenge and informs WD of the result.

Nevertheless, as pointed out in the WEP standard [23] sending both the challenge and its encrypted version may produce a security problem. This is due to the fact that the encryption algorithm used (RC4) is a stream cipher which generates a pseudorandom sequence and XORs it to the plaintext in order to create the ciphertext. Therefore, if we know the ciphertext and the plaintext, we can obtain the keystream straightaway. Nevertheless, in the standard it was only advised (but not required) to change the key and/or IV (Initialization Vector) frequently. Therefore, as observed in [10] the fact of not being required to change the key/IV, indirectly forces every receiver to accept repeated key/IV's, or risk otherwise not being compatible with some WEP compliant devices. That allows an attacker to successfully impersonate any station after having observed one single authentication. In [10] they call this attack *Authentication spoofing*.

Let us now forget for a moment that this flaw is known, suppose that we are asked to verify the security of WEP-SKA, and that it successfully passes the first part of our methodology. Then we would have to face the procedural verification of the protocol. The steps defined in Sec. 3.2 are as follows:

Step 6. Protocol pseudocode. The pseudocode for the WEP-SKA protocol is given in Listing 3.

```
Process WD:
    send(MA,AUTH,(id, 'shared key',1));
    receive(MA,AUTH,('shared key',2,info,result));
    if result == sucessful then
        chal = info;
        send(MA,AUTH,('shared key',3,$\{chal\}_k$));
    fi
    receive(MA,AUTH,('shared key',4,result));

Process AP:
    receive(MA,AUTH,(id,'shared key',1));
    if successful then
        result = sucessful;
        chal = new pseudorandom;
        send(MA,AUTH,('shared key',2,chal,result));
        receive(MA,AUTH,('shared key',3,$\{chal2\}_k$));
        if $\{chal2\}_k$ == $\{chal\}_k$ then
            send(MA,AUTH,('shared key',4,sucessful))
        fi
    fi
```

Listing 3: Pseudocode for the WEP-SKA procedures. WD stands from the Wireless Device to be authenticated, AP stands from Acces Point and k is the shared key. For simplicity, it is only shown the pseudocode for a successful execution. The field *info* conveys information dependent on the authentication algorithm, and *result* stores the result of the requested authentication. *MA, AUTH, 'shared key'* and the numbers used in the messages are fields specified in the standard, see [23]

Step 7. Formalization of the protocol. ProVerif has a problem here, because it does not support the XOR opperation[6].Nevertheless, since we don't need to apply complex derivation rules, we can get around this problem with a simple reduction rule simulating the XOR. This workaround can be seen in the code of the program available in [14], and is a rule telling that if the attacker knows both the challenge c, and the challenge c encrypted (i.e., xored) with the key k, then he can apply the xor function to recover the key k.

Step 8. Procedural verification. Once we have formalized the protocol, we run the tool to see if it succeeds or fails. In our case, running ProVerif with the code in [14], we observe a trace like the one shown in Listing 4.

[6]Although there are approaches for XOR-aware modifications of ProVerif. See [26].

WEP-SKA between legitimate WD_1 and AP:
$WD_1 \rightarrow AP \ : \ WD_1$
$AP \rightarrow WD_1 \ : \ Chall_1$
$WD_1 \rightarrow AP \ : \ \{Chall_1\}_k$
Attacker : $k = Chall_1 \oplus \{Chall_1\}_k \implies$ Attacker gains k
$AP \rightarrow WD_1$: OK
WEP-SKA between illegitimate $fake_{WD_2}$ and AP:
$fake_{WD_2} \rightarrow AP: \ fake_{WD_2}$
$AP \rightarrow fake_{WD_2} \ : \ Chall_2$
$fake_{WD_2} \rightarrow AP: \ \{Chall_2\}_k$
$AP \rightarrow fake_{WD_2}$: OK

Listing 4: An example attack trace found by ProVerif for the Authentication spoofing attack over WEP-SKA introduced in [10]. In the first block, the attacker eavesdrops on a WEP-SKA run between a legitimate WD and the AP. As a result, the attacker obtains the preshared key k. In the second block, the attacker uses the key k in order to successfully complete a WEP-SKA execution with the AP.

Since we have found an attack during the procedural verification with ProVerif, the requirements are not fulfilled. Therefore, we must go back to the first stage of the methodology once we have checked that no *coding* error has been made and redesign the protocol to avoid this attack. For instance, we could proceed like the authors of [10] suggest, which is to disallow the reuse of IVs in order to avoid this attack.

4.3 Case study III: complete verification

Now we shortly show how the full methodology is applied to the analysis of the protocol that was informally presented in [16], and whose formal security verification is performed in [15]. While applying our methodology, we found flaws in the design of the protocol that could lead to, for instance, replay attacks. Here we will just summarize the procedure followed (for further details, see [15]). The protocol is called CHAT-SRP (CHAos based Tickets - Secure Registration Protocol), and it is intended to be used in registration protocols for interactive platforms. The typical registration method used in these platforms requires the new users to provide an email address, verifying their identity when they access a link included in an email sent to the provided address[7]. This approach, although very user-friendly, is very insecure, because emails are almost always sent unprotected [18]. Therefore, an attacker may impersonate a user just by eavesdropping the activation email. CHAT-SRP generates a ticket (basically, a pseudo-random number), links it to the requesting user email, and sends it encrypted via HTTPS. The user, who is initially identified as being the owner of a mobile number (which we assume to be previously known and verified), besides accessing the activation link sent by email, has to provide the right ticket in order to validate his new account. This way we confirm that who

[7]This is known as EBIA, [19].

requested the registration is the one completing it. Once confirmed his identity, the protocol sends the new user a digital identity, which he could later use for performing robust cryptographic operations. Let us use our methodology to analyze it.

Step 1. Goals of the protocol. We want to register new users, providing them with new virtual identities to be used in the platform. Therefore, our more formally stated goals are:

1. Authenticate a user requesting registration.
2. Provide him/her a digital identity, keeping the identity's confidentiality and guaranteeing its authenticity (i.e., that it has been generated by a suitable Certification Authority).

Step 2. Attacker capabilities and security model. We assume the typical of the Dolev-Yao attacker. Note that this includes the capability of eavesdropping and blocking emails, which, as we said before, is our main concern. Besides, we also assume that our attacker can search in the Internet for the necessary information to start a registration process (in this case, a username and an email). Thus, our +/- capabilities list is shown in Listing 5.

```
        + Knows all the information required for registrationfor every user subject to
be registered (username and email).
```

Listing 5: +/- capabilities list for the Dolev-Yao attacker for the CHAT-SRP protocol.

Step 3. Design candidate. A sequence diagram for the protocol informally described at the beginning of this subsection is shown in Fig. 4, and we will use it as our design candidate. Keep in mind that, for brevity, we only show the result of several iterations of the methodology. Also, in this diagram and in the subsequent analysis, WS stands for Web Server, RA for Registration Authority and CA for Certification Authority. The three of them are assumed to be trusted authorities.

Step 4. Trust requirements. Applying Algorithm 1 to the elements in diagram in Fig. 4, we obtain the trust requirements in Listing 6.

```
        Step 1. username : None
                user@email.dom : None
        Step 2. code : Uniqueness, authenticity, confidentiality
        Step 3. username : None
                code : Uniqueness, authenticity, confidentiality
        Step 4. username : None
                code : Uniqueness, authenticity, confidentiality
        Step 5. Ticket Request : Authenticity, confidentiality
        Step 6. ticket : Authenticity, confidentiality, uniqueness
        Step 7. ticket : Authenticity, confidentiality, uniqueness
        Step 8. link : Authenticity, confidentiality, uniqueness
```

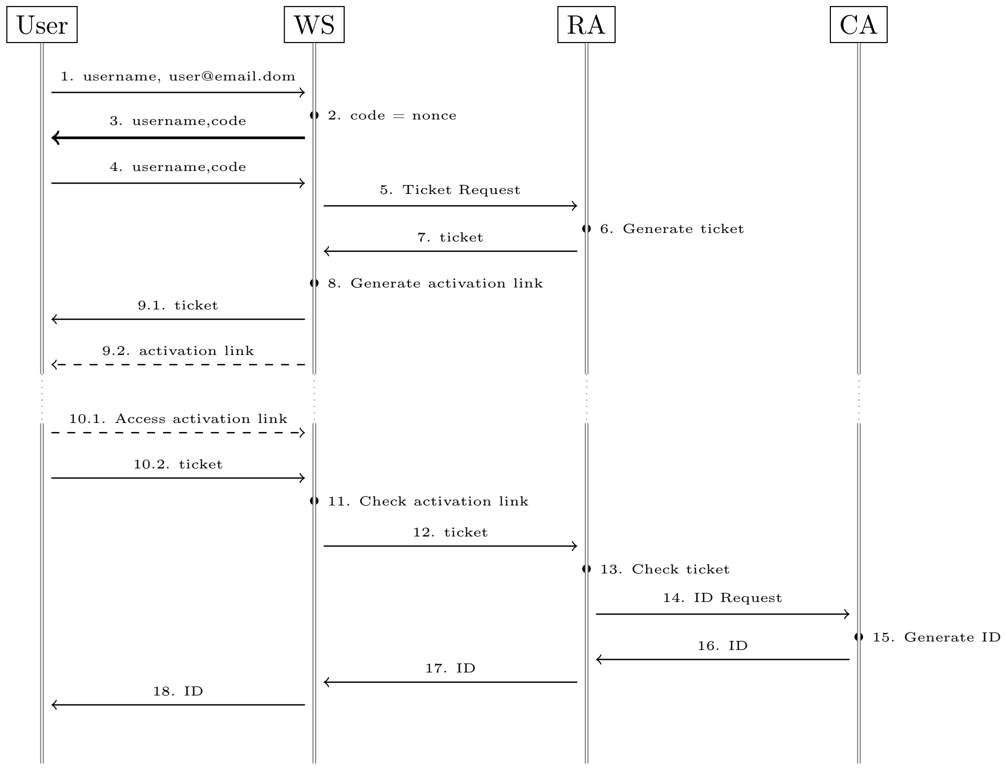


Figure 4: Sequence diagram of the protocol messages. The dashed lines represent unprotected communications; the thin continuous ones represent SSL protected communications, and the thick continuous line represents the message sent using the extra authenticated channel (SMS). Although not shown in the diagram for readability, the ticket is generated as the hash of a nonce and the user's email.

```
Step 9.1. ticket : Authenticity, confidentiality, uniqueness
Step 9.2. link : Authenticity, uniqueness
Step 10.1. link : None
Step 10.2. ticket : Authenticity, confidentiality, uniqueness
Step 11. Nothing to do here
Step 12. ticket : Authenticity, confidentiality, uniqueness
Step 13. Nothing to do here
Step 14. ID Request : Authenticity, confidentiality
Step 15. ID : Authenticity, confidentiality, uniqueness
Step 16. ID : Authenticity, confidentiality, uniqueness
Step 17. ID : Authenticity, confidentiality, uniqueness
Step 18. ID : Authenticity, confidentiality, uniqueness
```

Listing 6: List of trust requirements for CHAT-SRP.

Basically, at steps 1, 3 and 4, we are considering the extra capability added to our attacker in our +/- capabilities list. Also, note that the activation link, when created in step 8 is required authenticity and uniqueness (the confidentiality requirement is just a consequence of having just been created). However, in step 9.2 it is sent via email (an insecure channel), thus confidentiality cannot be assumed. Moreover, the link is sent at step 10.2 (which simulates the access to the corresponding web site) cannot be assumed to be confidential, nor unique, nor authenticated, since it can be originated from anywhere. The ticket and ID are expected to be kept authenticated, secret and unique, since they are always conveyed through SSL-secured channels.

Since we have required the protocol to provide secrecy, authenticity and uniqueness, we decided to use ProVerif, which allows to check all the stated properties.

Step 5. Informal verification. Going through all the steps in Listing 6 we do not find any inconsistency with the attacker capabilities nor the +/- capabilities list specified before. Moreover, we have taken into account the attacker's capability to know all the personal information required to initiate a registration request (since we have place no trust requirement in the username and email elements in steps 1, 3 and 4). For the remaining steps, the required trust properties are consistent with the properties of the channels used to conveyed the associated elements (we must remark here that the WS, RA and CA are trusted authorities). Thus, we can continue, considering that our design successfully passes our informal verification step.

Step 6. Protocol seudocode. For brevity, we do not include the pseudocode here. Instead, it is accessible from [13]. Note that, again, it was obtained after several iterations of our methodology, in which we detected and corrected security flaws.

Step 7. Formalization of the protocol. In step 4 we decided to use ProVerif for verification. From the pseudocode, it is not hard to obtain a formalization for ProVerif. For further details on the formalization of the protocol, see [15] or the pseudocode and formalization for ProVerif in [13].

Step 8. Procedural verification. Running ProVerif on the code in [13] shows no violation on the security requirements. Therefore, we can assume that they have been accomplished.

Therefore, following our methodology, as we have briefly shown, CHAT-SRP is seen to fulfill our secrecy, authenticity and uniqueness requirements.

5 Conclusion

In this work we have designed a methodology for verifying security properties of communication protocols. It is mainly divided in two parts, the first one devoted to a preliminary study of the properties of the context where the protocol will be applied, its security requirements and an informal verification. The second part is dedicated to a procedural verification of these security requirements, applying well-known and widely accepted tools and/or procedures. By adopting an iterative methodology in which the early processes are less time consuming, we avoid wasting resources by increasing the probability of detecting basic flaws at the beginning of the analysis. We also provide three examples in order to help to understand which are the advantages of applying this methodology. These examples (MANA III, WEP-SKA and CHAT-SRP) are representative, since they are real protocols that have been presented to the community, and specially in the case of WEP-SKA, widely used. Note that the first two examples study already known security flaws. However, our aim is not to detect new flaws, but to show how our methodology can be applied and the benefits of doing so.

We would like to remark that, when designing and evaluating communication protocols, the application of such a methodology helps to detect and avoid flaws that can lead to attacks on the protocols, like we have shown. It is important to note that both phases of the methodology are required: if the first phase is skipped, a wrong contextualization may render the formalization invalid; if, in the other hand, we skip the second phase, most probably we will not realize several involved nuances that, in turn, can lead to security flaws. By using what we called procedural methods, like formal protocol verifiers, or the computational model for the verification of protocols, we can prevent much of the damage caused by flawed designs. Nevertheless, the human factor also intervenes in the procedural analysis of protocols, and a wrong formalization can make some flaws to pass unnoticed. Hence, like in every engineering process, this does not provide a 100% success rate, but it does help to avoid many flaws.

References

[1] Common criteria for information technology security evaluation – part 3: Security assurance components. Technical report, July 2009.

[2] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS*, pages 3–22, 2000.

[3] Ross J. Anderson and Roger M. Needham. Robustness principles for public key protocols. In *CRYPTO*, pages 236–247, 1995.

[4] Christian Haack Alan Jeffrey Andrew D. Gordon. Cryptyc: cryptographic protocol type checker. `http://cryptyc.cs.depaul.edu/`, 2002.

[5] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL*, pages 90–101, 2009.

[6] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT*, pages 259–274, 2000.

[7] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In *POPL*, pages 445–456, 2010.

[8] Bruno Blanchet. *ProVerif Automatic Cryptographic Protocol Verifier User Manual*. CNRS, Département d'Informatique École Normale Supérieure, Paris, July 2010.

[9] Bruno Blanchet and David Cadé. Cryptoverif: Cryptographic protocol verifier in the computational model. `http://www.cryptoverif.ens.fr/`, 2012.

[10] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001.

[11] Sagar Chaki and Anupam Datta. Aspier: An automated framework for verifying security protocol implementations. In *CSF*, pages 172–185, 2009.

[12] Università di Genova, CASSIS group, Information Security Group (ETHZ), and Siemens AG. Avispa: Automated validation of internet security protocols and applications, 2006.

[13] Jesus Diaz, David Arroyo, and F.B. Rodriguez. Code of chat-srp for proverif. `http://www.ii.uam.es/~gnb/chat-srp.pv`, 2012.

[14] Jesus Diaz, David Arroyo, and F.B. Rodriguez. Code of wep-ska for proverif. `http://www.ii.uam.es/~gnb/wep-auth-spoofing.pv`, 2012.

[15] Jesus Diaz, David Arroyo, and F.B. Rodriguez. Formal security analysis of registration protocols for interactive systems: a methodology and a case of study. *submitted*, 2012.

[16] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. An approach for adapting moodle into a secure infrastructure. In *CISIS*, pages 214–221, 2011.

[17] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[18] Stephen Farrell. Why don't we encrypt our email? *IEEE Internet Computing*, 13(1):82–85, 2009.

[19] Simson L. Garfinkel. Email-based identification and authentication: An alternative to pki? *IEEE Security & Privacy*, 1(6):20–26, 2003.

[20] Christian Gehrmann, Chris J. Mitchell, and Kaisa Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7:2004, 2004.

[21] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real c code. In *VMCAI*, pages 363–379, 2005.

[22] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover security design flaws using the stride approach. `http://msdn.microsoft.com/en-us/magazine/cc163519.aspx`, 2006.

[23] IEEE. Wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE Standard 802.11, June 1999.

[24] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. Technical Report 19, Verimag Technical Report, 2004.

[25] Jan Jurjens. *Secure Systems Development with UML*. SpringerVerlag, 2003.

[26] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. In *ACM Conference on Computer and Communications Security*, pages 129–138, 2008.

[27] Shin'ichiro Matsuo, Kunihiko Miyazaki, Akira Otsuka, and David A. Basin. How to evaluate the security of real-life cryptographic protocols? - the cases of iso/iec 29128 and cryptrec. In *Financial Cryptography Workshops*, pages 182–194, 2010.

[28] Larry Paulson, Tobias Nipkow, and Makarius Wenzel. Isabelle, 2012.

[29] Mikko T. Siponen and Robert Willison. Information security management standards: Problems and solutions. *Information & Management*, 46(5):267–270, 2009.

[30] Bomil Suh and Ingoo Han. The is risk analysis based on a business model. *Information & Management*, 41(2):149–158, 2003.

[31] Scott Swigart and Sean Campbell. Sdl series – article #1: Investigating the security development lifecycle at microsoft. `http://www.microsoft.com/security/sdl/resources/publications.aspx`, October 2008.

[32] Ford-Long Wong and Frank Stajano. Multi-channel protocols. In *Security Protocols Workshop*, pages 112–127, 2005.

[33] Quey-Jen Yeh and Arthur Jung-Ting Chang. Threats and countermeasures for information system security: A cross-industry study. *Information & Management*, 44(5):480–491, 2007.