Verification with AVISPA to Engineer Network Security Protocols

Florian Kammüller, Glenford Mapp, Sandip Patel, and Abubaker Sadiq Sani
Middlesex University
Computer Communications Group
f.kammueller@mdx.ac.uk, mapp@mdx.a.cuk, sp1264@live.mdx.ac.uk, ss1234@live.mdx.ac.uk

***Abstract*—This paper summarizes work on formal mechanized verification of security protocols using Avispa, a model checker dedicated to security protocols. Avispa has been successfully used in various Master's projects. In this paper, we present two outstanding projects of quite different nature that highlight the spectrum of formal security protocol verification and lead us to a proposition of engineering practice for the development of secure protocols based on two main ideas (a) refactoring existing formalisations to prove adaptations of security protocols (b) compositional proof of new protocols allowing the combination and reuse of (parts of) existing formalisations of other protocols. This paper presents first Radius-SHA256, an adaptation of the Radius protocol for remote authentication for network access to the secure hash function SHA-256. Second, we present the Secure Simple Protocol which is an extension for security of a protocol developed at our university for next generation networks. Both protocols have been formalized in the Avispa model checker and security has been proved.**

***Keywords*-Security protocols, Model Checking, Cryptographic Hashes, Simple Protocol**

I. Introduction

Radius [19], [20], a remote authentication protocol used for building up secure communications of clients with networks via network access servers, uses the message digest function MD5, a hash function which has meanwhile been proven to have security weaknesses. By contrast, the hash function SHA-256 still remains unchallenged. Although seemingly straightforward and thus tempting, simply replacing MD5 by SHA-256 in the Radius protocol must be considered potentially harmful since authentication protocols are extremely sensitive to minor changes as the history of attacks shows. In December 2008, an attack on the SSL protocol has been demonstrated based on the previously discovered collisions of the MD5 hash function [16]. The engineers of that attack recommend the discontinuation of use of SSL based on MD5. Fortunately, for SSL the use of the hash function is already by design a choice point. For Radius, this flexibility is not yet established; this is the subject and one of the results of this paper. Triggered by the alarming history of attacks of security protocols, formal verification techniques have long been deemed to be a way out. We investigate whether Radius-SHA256 – our proposed adaptation of the Radius protocol – can provide better security guarantees than its original. To provide evidence based on mathematical rigor we use the Avispa model checker. Fortunately, we can rely on the rich data base of this tool providing a model of the original protocol. By adapting this model to our Radius-SHA256 and checking that the original security guarantees still hold we prove two things (a) that Radius-SHA256 is secure and (b) that the security guarantees have general validity, i.e. they can be carried over to protocols Radius-X for hashes X. The latter result corresponds to a reduction of Radius security to the security of the underlying hash function.

Model checking, a push-button technology for mathematical verification of finite state systems has been discovered to be a suitable tool for security analysis of authentication protocols, e.g. [7]. Ever since, this technology has proved to be useful for the engineering of secure protocols, e.g. for adaptation of the Kerberos protocols to mobile scenarios [6]. However, little attention has been given to investigate to what extent we can use known engineering techniques, like refactoring, reuse, and composition to help us engineer formal security verifications of protocols. This paper is to be seen as a first step towards such an engineering process. We mainly present two distinct and unrelated case studies on Avispa formalisations. The first one being the aforementioned Radius and the second one a new specially LAN-centric transport protocol called simple protocol (SP) developed in our research group [21] and extended here by security, i.e. authentication. As a second engineering exercise, we report on this secured version of the SP protocol. This exercise shows how a new development of a special purpose protocol can profit from a simultaneous modelling and analysis with a dedicated modelchecker like Avispa. The two case studies need not be related since they just serve as case studies for engineering security protocols with general engineering principles like refactoring, reuse and composition. Even though, there is a bridge between those seemingly unrelated projects: securing local servers and services. So the first project (Radius) looks at a server protocol and the other project (SP) looks at a secure, optimized and tunable protocol for local servers.

This paper is an extension of the conference paper [4] and is based on the Masters Theses of two of the authors [11], [14]. The extensions are a more verbose introduction to the SP protocol and the technical presentation of the Avispa encoding of SP. In this paper we first provide the prerequisites of this project: brief introductions to the

Radius protocol, the Simple Protocol, Avispa model checking, and hashes (Section II). From there, we develop our new version Radius-SHA256 by introducing its model in Avispa in detail (Section III) and illustrate how this model can be efficiently used to verify security goals (Section III-D). To illustrate that Modelchecking is also useful in the engineering of new protocols we show its application to the Simple Protocol (Section IV). We first give a motivation and deeper introduction to this protocol and its context for future networks thereby extending the original paper [4]. Next we show how this Simple Protocol can be extended step by step introducing cryptographic keys to add authentication and secure it. We finally offer conclusions and an outlook (Section V).

II. Background

A. Radius

One of the major issues with networks is their security and one response to this challenge are authentication protocols. Radius is a popular protocol providing security to communication channels. Radius stands for *Remote Authentication Dial in User Service* and serves to secure communication between *Network Access Servers* (NAS) and so-called Radius servers. Radius satisfies the AAA (Authentication, Authorization and Accounting) protocol standards in both local and roaming situations. In January 1997, Radius standards were first introduced in RFC 2058 and Radius accounting in RFC 2059. After that RFC 2138 and RFC 2139 were published and they made the previous RFC obsolete. They both were made obsolete in turn by RFC 2865 and RFC 2866 respectively. Following were updates by RFCs 2868, 3575, and 5080 [20].

Assume that there is an Internet service provider (ISP) and he has two NAS. A NAS allows a user to connect directly to the ISP's network and be accepted by a core router which directly connect with ISP's network backbone. When a user wants to access his services, he sends a request to the NAS which forwards the user request to the main server to check the supplied credentials. This process is called authentication.

After authentication, the NAS has to check the access list of the user and then decide which services are permitted to this user. The RADIUS server then replies to the NAS with Access Reject, Access Challenge, or Access Accept as illustrated in Figure 1. This information is forwarded by the Radius server to the NAS. This is called authorization. Once a user is authenticated and authorized successfully, the NAS creates a connection between the user and the main server through which both can exchange their information. This secure connection is called a session. All the information regarding the session will be saved by the NAS for its accounting purposes. It includes start time of session, termination time of session, size of total received and sent data, amongst other information for accounting.

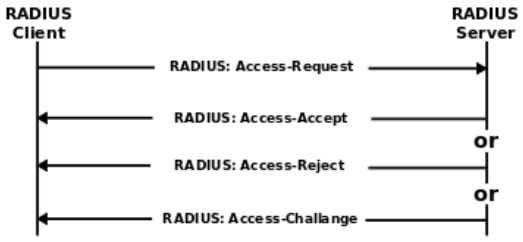

Figure 1. Radius access request and possible replies [18]

B. Simple Protocol

A new trend in next generation networks is the divergence between local area networks (LAN) and wide area networks (WAN) because there is still an increase of efficiency to be expected in LANs. Additionally, the ubiquity of computing devices and common usage of mobile devices asks for a flexibility that is better supported with fixed core networks and flexible wireless networks at the periphery. A reconsideration of the TCP/IP seems appropriate since adaptation of TCP to the often heterogeneous requirements of local wireless networks is not easy. The Simple Protocol (SP) [10] is intended to be used in combination with TCP but TCP for the WAN and SP for the LAN communication. SP is part of a wider development of the Y-Comm framework [21] – a new architecture for mobile heterogeneous networking.

A specially LAN-centric transport protocol has different requirements from a WAN transport protocol, e.g. TCP, since performance issues differ. These requirements mark the design decision that define SP [10]. Since most LAN communications consist of messages or transactions, SP supports a message-based communication in contrast to TCP streams. The higher speed available in LAN is exploited by using a larger window size for SP than WAN protocols: SP supports 4MB message sizes by default and can even be increased. In order to keep packet processing simple, SP uses a small number of connection states as well as packet types. Flexibility is achieved by allowing Quality of Service (QoS) to be set using the packet types.

In this paper (Section IV), we summarize briefly how the Avispa support helped in designing a secure extension of SP by hybrid cryptography. Extending the initial connection part of the protocol, we add public-key based authentication while simultaneously exchanging symmetric session keys for the following secured data exchange of SP. The protocol achieves authenticity by public keys while preserving its efficiency to an extent through the use of faster symmetric key encryption.

C. Avispa

Avispa stands for Automated Validation of Internet Security-sensitive Protocols and Applications [2]. To model and analyze a protocol, Avispa provides its own High-Level Protocol Specification Language (HLSPL). In order to check

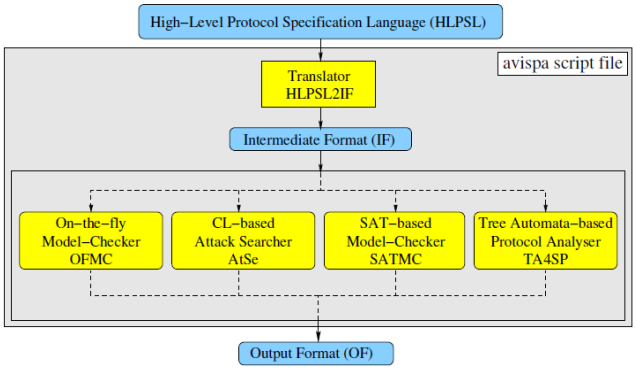

Figure 2. Avispa: language formats and tool architecture [2]

security, Avispa translates the given HLSPL specification in the intermediate format IF, which is then the basis for four different verification machines that can be applied to model check security properties on a protocol expressed as depicted in Figure 2. Avispa uses Dolev-Yao channels annotating them as a type as `channel(dy)`. This means that the attacker is assumed to be able to do eavesdropping, intercepting and faking on these channels. Protocols can be very naturally specified in Avispa using the *role* concept. Every principal is modeled as such a role which enables encapsulating its communication parameters, local variables and constants. Based on that, a role describes state changes by defining transitions between states that may depend on pre- and postconditions of the current state. Roles can furthermore be instantiated in other roles. This enables the composition of the single roles representing the single principals into a protocol session while synchronizing them on their communication. It also enables specifying an attacker. Once the protocol is thus specified predefined HLSPL propositions, most prominently secrecy and authentication can be automatically verified. We introduce more detail on HLSPL constructs, their IF translation, and the verification features when applying them to formalize Radius-SHA256 in the following section.

D. Hash Functions

Hash functions – also known as message digests or compression functions – map arbitrary length inputs to fixed size outputs. They are considered as cryptographic hash functions if they provide the following three properties: (a) they cannot be inverted, i.e. given $y = H(x)$ the input x cannot be found, (b) it is impossible to find collisions, i.e. we cannot find x, y with $H(x) = H(y)$, and (c) given an input hash pair it is impossible to find another input with the same hash value, i.e. for $H(x) = y$ we cannot find x' such that $H(x') = y$. The latter two properties resemble each other expressing the idea of *collision resistance* but the second one is stronger.

These basic properties of good hashes give rise to use them for cryptography. However, since a hash is a deterministic function it has as such not the same quality as an encryption algorithm: anyone can apply the hash. However, a hash can be easily combined with a shared secret to provide authentication which is often used for so-called message authentication codes (MAC). For example, let Kcs be a shared secret. Then, $H(Kcs)$ can be used as an authentication token since only principals who have access to Kcs can produce this token.

III. RADIUS-SHA256

In this section, we present the protocol Radius-SHA256 as derived from the classical Radius of RFC2865/66 by replacing MD5 by SHA-256. At the abstract protocol level this replacement seems simple but in order to ensure that this change of the original protocol preserves the security properties, we start from the formal presentation of the original Radius protocol and develop the new Radius-SHA256 on that formal basis. This enforces a detailed investigation of the necessary adjustment to the old – no longer secure – version of Radius and in addition enables comparison to the previously established security guarantees showing whether they still hold. From an engineering perspective, this procedure corresponds to a kind of refactoring of a protocol specification: re-engineering the previous security specification enables re-invocation of the previous verification by rerunning security check routines.

We introduce the protocol Radius-SHA256 by its formal model in HLSPL, the specification language of Avispa. Its level of abstraction is sufficient to comprehend just the major gist of the protocol. This model contains four roles: client, server, session, and environment. The idea is that the client role represents the NAS and the server role represents the Radius server. In applications, client and server might as well be represented by proxies depending on the type of network. For the formal presentation of the protocol we simplify by summarizing the scenario as a client-server session. As a session we consider the time period of a client-server communication. The attacker is modeled by the role of the environment that specifies the basis for the attacks on protocol executions.

Each of these components client, server, session and environment is modeled by a so-called "role" in HLSPL. Client (Section III-A) and server (Section III-B) define the two matching sides of the protocol; their composition as defined in the role session only gives the full protocol (see Section III-C and Figure 5) which can again be instantiated to model legal session and attacker.

A. Client-side Protocol

The client role is specified in Figure 3. This role definition defines the protocol by specifying the necessary entities, like identifiers, messages and used cryptographic primitives, e.g. the symmetric key `Kcs` in its header. Note, here how we

```
role client(C,S: agent,
   Kcs: symmetric_key,
   SHA256: hash_func,
   Success, Failure: text,
   Access_accept,Access_reject: text,
   SND, RCV: channel(dy))
played_by C def=
local  State: nat,
   NAS_ID, NAS_Port: text,
   Chall_Message: text
const  kcs: protocol_id,
   sec_c_Kcs : protocol_id
init State := 0
transition
   t1. State  = 0 ∧ RCV(start) ⇒
       State':= 1 ∧ NAS_ID':=new()
       ∧ NAS_Port':=new()
       ∧ SND(NAS_ID'.NAS_Port'.SHA256(Kcs))
       ∧ secret(Kcs,sec_c_Kcs,C,S)
   t2. State  = 1 ∧ RCV(NAS_ID.Access_accept) ⇒
       State':= 2 ∧ SND(NAS_ID.Success)
   t3. State  = 1 ∧ RCV(NAS_ID.Access_reject) ⇒
       State':= 3 ∧ SND(NAS_ID.Failure)
   t4. State  = 1 ∧ RCV(NAS_ID.Chall_Message') ⇒
       State':= 4 ∧ SND(NAS_ID.Chall_Message'_Kcs)
                  ∧ witness(C,S,kcs,Kcs)
   t5. State  = 4 ∧ RCV(NAS_ID.Access_accept) ⇒
       State':= 5 ∧ SND(NAS_ID.Success)
end role
```

Figure 3. Client role of Radius-SHA256 in HLSPL

define SHA256 to be a *hash function* in this header by using the Avispa keyword hash_func. This function is applied in the first transition of the following client-side of the protocol specification. In detail the steps of the protocol are defined as *state transitions* that are conditional on logical conditions of a current state State $\in \{1, \ldots, 5\}$: each of the five rules in the transition section in Figure 3 defines a *precondition* for this current state (to the left of the implication arrow $\Rightarrow$) and a *postcondition* on the post state State' of a transition after the $\Rightarrow$. The conditions are conjoined by logical conjunction with $\wedge$. The initial state is State zero. For example, the first transition t1 in Figure 3 can be read as follows. If the precondition holds, i.e. the current state is "state 0" and the role receives on its input channel RCV the message start, then the transition t1 is enabled. If this transitions fires, the post-state is "state 1" and the message NAS_ID.Success is sent on the output channel SND. The following transitions can be read in the same manner. Since the client represents only one principal in this protocol, we need to need to define the server side of the protocol to complement it.

B. Server-side Protocol

Figure 4 now shows the definition of the second principal in the model of Radius-SHA256: the Radius-server. The transitions defined in the role server correspond to the transitions of the client. Each SND on one side corresponds to a RCV on the other side. However, in order to put

```
role server(C,S: agent,
            Kcs: symmetric_key,
            SHA256: hash_func,
            Success, Failure: text,
            Access_accept,Access_reject: text,
            SND, RCV: channel(dy))
played_by S def=
local State: nat,
      NAS_ID, NAS_Port : text,
      Chall_Message     : text
const kcs: protocol_id,
      sec_s_Kcs : protocol_id
init State := 11
transition
t1. State  = 11
    ∧ RCV(NAS_ID'.NAS_Port'.SHA256(Kcs)) ⇒
    State':= 12 ∧ SND(NAS_ID'.Access_accept)
                ∧ secret(Kcs,sec_s_Kcs,C,S)
t2. State  = 12 ∧ RCV(NAS_ID.Success) ⇒
    State':= 13
t3. State  = 11
    ∧ RCV(NAS_ID'.NAS_Port'.SHA256(Kcs)) ⇒
    State':= 14 ∧ SND(NAS_ID'.Access_reject)
t4. State = 14 ∧ RCV(NAS_ID.Failure) ⇒
    State':= 15
t5. State  = 11
    ∧ RCV(NAS_ID'.NAS_Port'.SHA256(Kcs)) ⇒
    State':= 16 ∧ Chall_Message':=new()
                ∧ SND(NAS_ID'.Chall_Message')
t6. State  = 16 ∧ RCV(NAS_ID.Chall_Message_Kcs) ⇒
    State':= 17 ∧ SND(NAS_ID.Access_accept)
                     ∧ request(S,C,kcs,Kcs)
t7. State  = 17 ∧ RCV(NAS_ID.Success) ⇒
    State':= 18
end role
```

Figure 4. Server role of Radius-SHA256 in HLSPL

these building blocks together, we first have to define the composition. This is done in a further role for the session, presented in the following section.

C. Session and Attacker

The two roles of client and server are combined by defining a role for the session. Session uses the composition keyword to couple the two instances of client and server synchronized by common parameters.

```
role session(C,S: agent,
             Kcs: symmetric_key,
             SHA256: hash_func,
             Success, Failure: text,
             Access_accept,Access_reject: text) def=
local
  S1, S2 : channel (dy),
  R1, R2 : channel (dy)
composition
  client(C,S,Kcs,SHA256,Success,Failure,
         Access_accept,Access_reject,S1,R1) ∧
  server(C,S,Kcs,SHA256,Success,Failure,
         Access_accept,Access_reject,S2,R2)
end role
```

The synchronization couples the transitions of the client with the server over their connecting channels. For example, the

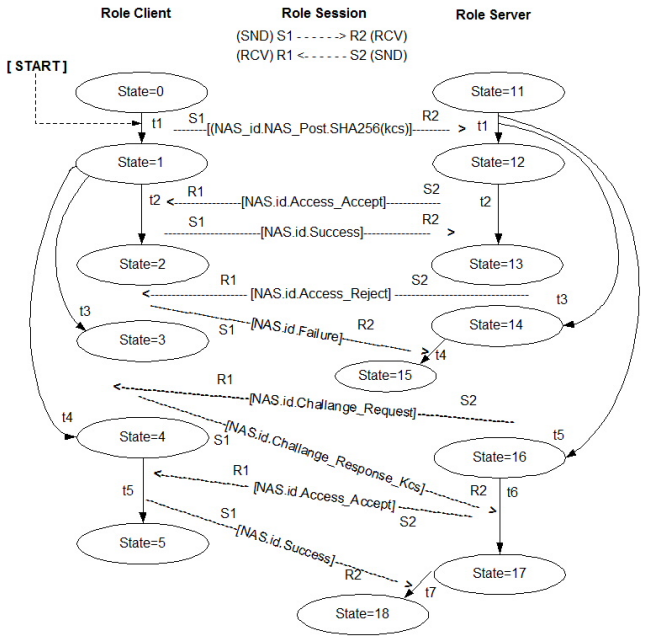

Figure 5. Composition of client and server yields protocol.

message `SND(NAS_ID.Success)` of `t2` in client is now being sent over `S1` and coupled via `R2` to the message `RCV(NAS_ID.Success)` of server. The composition that is defined in the role session actually defines the protocol between the roles client (Section III-A) and server (Section III-B) by instantiating their channels such that they mutually connect; the overall protocol is best illustrated graphically (see Figure 5).

The environment represents the attacker and uses a composition, now in turn of two session instances, where the first is one between two agents `c1` and `s1` and the second generalizes the first agent to be `i` – an unspecified agent that triggers the search for intruder possibilities incorporating agents. Note, also that the `SHA256` is given openly to the environment signifying that the attacker knows it and can use it which formalizes the idea of a hash that it is applicable by everyone (see Section II-D).

```
role environment() def=
const c1,s1: agent,
      sha256: hash_func,
      succs, fails: text,
      acc_acp, acc_rej: text,
      kcsk, kisk, kcik: symmetric_key,
      kcs: protocol_id
intruder_knowledge = {c1,s1,sha256,kisk,kcik,
                      succs, fails, acc_acp, acc_rej}
composition
session(c1,s1,kcsk,sha256,succs,fails,acc_acp,acc_rej)
∧
session(i, s1,kisk,sha256,succs,fails,acc_acp,acc_rej)
end role
```

The representation is abstract enough to be comprehensible while being in places a bit superficial. We dig deeper down into the lower levels of the Avispa model in the next section to investigate the influence of the hash function on the Radius-SHA256.

D. Security Verification

This section now illustrates how the actual model checking process of the Avispa tool automatically translates the high level protocol model in HLSPL defined in the previous section and performs a complete state analysis over the resulting internal Kripke structure representing this model. The verification is relative to a set of security properties specifying the goals of the authentication that we will illustrate first.

E. Security Properties and Verification Process

Given the implementation of the protocol as described in the previous section, we can now use the inbuilt features of Avispa to verify security in a push-button manner. Avispa provides two features for protocol verification: secrecy of keys and authentication. The secrecy of the server and client keys and authentication of client and server are given as verification commands to Avispa as follows.

```
goal
  secrecy_of sec_c_Kcs, sec_s_Kcs
  authentication_on kcs
end goal
```

The meaning of these two formulas can be illustrated more closely by inspecting their translation into the IF format. We apply all four back-ends OFMC, CLAtSE, SATMC, and TA4SP of the Avispa tool to the Radius-256 specification. For the full IF representation and the performance details of the analysis see [11]. The main observation is that the original security guarantees shown for Radius can be carried over to the protocol Radius-SHA256 by *simply replacing* the hash function MD5 by SHA-256 in the specification. The above secrecy and authentication properties verify just the same.

To understand the effect that the choice of a particular hash function, i.e. MD5, SHA-256, or any other cryptographic hash function has on the security guarantees, we need to inspect the IF version in more detail. First of all, a hash function application in HLSPL like `SHA256(Kcs)` is translated into IF as `apply(SHA256,Kcs)`. According to the Avispa semantics [2], this `apply` operator is reserved for the application of hash functions which manifests itself in the following type.

```
apply(F,Arg) apply: message × message → message
```

However, there seems to be no further semantics attached to the type. The defining properties of a cryptographic hash function are provided implicitly by defining the intruder knowledge for hashes as follows.

```
step gen_apply (PreludeM1,PreludeM2) :=
     iknows(PreludeM1).iknows(PreludeM2) ⇒
     iknows(apply(PreludeM1,PreludeM2))
```

Since the apply-operator can produce a hash, the intruder can apply a hash himself but Avispa's intruder semantics provides no rule to inverse a hash function nor any rule enabling collision detection for the intruder.

F. Evaluation and Generalization

Wrapping up the discussed security verification we see that the verification of Radius-SHA256 yields exactly the same guarantees as the classical Radius of RFC 2865/6. In this final section, we show up the consequences of this mechanized verification.

Primarily, the re-engineered modelling and verification for the Radius-256 protocol in Avispa shows that the guarantees of secrecy of keys and mutual authentication that have already been shown for the classical Radius version MD5 equally hold for Radius-256.

Next, our construction process reveals that the exchange of MD5 by another hash function in the Avispa model is simply replacing one (presumably) secure cryptographic hash function by another. As we have observed in the previous section, the Avispa semantics of a hash models hash functions abstractly. Thus, we observe that the verification depends only on the general assumption that *some* hash function is used in the protocol. Therefore the derived result can be generalized to *all* secure hash functions.

Theorem 1: The Avispa guarantees of secrecy of keys and authentication of the Radius protocol hold for all secure cryptographic hash functions.

Note, however, that this verification does presuppose a secure hash function. That is, the proved result is not valid if the assumed cryptographic strength of the hash function is flawed, like in the case of MD5.

Since the Avispa model cannot cover the implicit part of the hash function security proof, the analysis does not reveal possible attacks. However, the aforementioned attack on SSL [16] could be used as a guideline to produce a similar attack on the classical Radius protocol based on MD5. On the other hand, the generalization presented in this paper is not trivial: its proof relies on the re-engineering of the Radius for SHA-256 and the observation that this re-engineering is applicable to any secure hash function.

IV. Secure Simple Protocol

A. Transport Protocols and Future Network Environments

We are witnessing an explosive change in terms of the different types of wireless networks being developed and deployed. Thus, future network environments will primarily consist of multiple local wireless networks, each with a different Quality-of-Service (QoS). Users will be always connected by switching between available networks using vertical handover techniques [8].

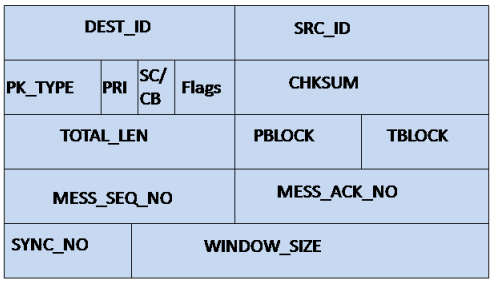

Figure 6. The structure of Ycomm/SP.

In this new world, a connection between two devices may therefore involve several network interfaces. This reality is beyond the scope of many TCP implementations and protocols such as SCTP [12] have been developed to replace TCP.

Another approach being explored is to keep TCP as a Wide Area transport protocol, but use another protocol to handle communication on local networks. Such an approach is being adopted by the Y-Comm Group [9], [21]. There are other factors favoring this approach: firstly, the common tack of tuning TCP to deal with different types of local networks has not been as effective as initially hoped. TCP is, in fact, a relatively complicated protocol as it needs to support different transport features including connection management, reliability, streaming and congestion evaluation and response. This means that it has been difficult for TCP to adjust to handover issues [3] without substantial support from device interfaces; we believe a local transport protocol should be simpler to use. Secondly, in order make use of higher speeds that are generally available in local environments, transport window sizes must be substantially larger by default. Thirdly there is a need to support different Qualities-of-Service is an explicit and flexible way. Finally, the issue of local area network security in terms ensuring that resources are properly balanced among several users must be urgently addressed to ensure improved user experience.

B. The Simple Protocol

The Simple Protocol (SP) [10] is being developed by the Y-Comm Group for Local area communications. The SP diagram is shown in Figure 6 with a brief explanation of the various fields shown in Figure 7.

As can be seen from Figures 6 and 7, SP is a message-based protocol where messages are broken down into a number of blocks. Unlike TCP, SP supports the concept of packet types and uses explicit phases: connection setup, data exchange and close; this allows the protocol to be quickly processed. It also supports a 4 MB window size, leading to improved transfer rates for large data exchanges. In SP,

- DEST_ID (16) – identifying remote end
- SRC_ID (16) – from source end
- PK_TYPE (4) - packet type
- PRI (2) - supports 4 priority levels
- SC (2) – destination scope
- CB (2) – supports ECN
- CHKSUM (16) – sixteen bit checksum
- TOTAL_LEN (16) – total packet length
- PBLOCK (8) – the present block
- TBLOCK (8) – the total number of blocks
- MESS_SEQ_NO (16) – last message sent
- MESS_ACK_NO (16) – last message received
- SYNC_NO (8) – the last ACK received
- WINDOW_SIZE (24) – the window size

Figure 7. Explaining what the fields mean.

acknowledgments can also be monitored leading to faster recovery times.

C. Support for Servers

Because SP is a message based protocol, it allows the application and the protocol stack to operate in a more asynchronous manner using an event-driven interface for receiving packets. Hence the interaction between the application and SP for reception revolves around the application being made aware of four key events:

1) A new connection has been established
2) A new message has been received on a connection
3) The connection has been reset by the network or other side
4) The other side has closed its connection, i.e. it is finished sending all its data.

Since it is possible to attach call back functions to each event, it is possible to build local servers that are totally event-driven. This leads to much more efficient server implementations. In addition, the application can tune SP so as to optimize the local environment. Thus a local transaction server can tune SP to suppress transport level acknowledgment requests since the reply by the server of the client's request will also be interpreted by the client transport system that the original request was correctly received. This too is a significant optimization for efficient server implementations. SP is therefore a powerful local protocol and hence the need to look at a secure version of SP.

D. Security Mechanism in SP

Because SP uses a connection phase, we can use this to improve the security of the system and to enhance communications. When connecting to servers, the concept of scope is supported. So a server can only be accessed using a scope which is defined by its functionality [1]. Four scopes are supported using 2 bits: Scope 00 means that the server can only be connected to processes on the same machine. Scope 01 means that the server can only be accessed by machines on the same LAN, while scope 10 indicates that access to the server is only accessible by machines on the same site while a scope of 11 shows that the server may be globally accessed. The connection phase of SP can be used to set up keys which are formed using local area parameters to ensure secure communication between machines in the local area.

The protocol SP consists of two parts: the connection part and the data transmission. The connection part establishes a communication between processes A and B to prepare a data transmission according to these established connection parameters. Thereby, a "connected" state is reached during which data may be transmitted before the connection is closed again. During data transmission, SP uses synchronization numbers (`SYNC_NO`) for each message and acknowledgments replying those message numbers to ensure safe transmission. This sequential message numbering can be used as well to secure the protocol against replay attacks, i.e. resending of previously intercepted messages by adversaries. However, to ensure this security, we need to keep the message numbers secret. To do that, we establish a session key in the connection part of SP. We assume that a global public key infrastructure provides certified identities, that is for every principal X on the network we have a signed pair $(K_X, X)_{K_C^{-1}}$ of a public key K_X associated to the principal's identity (for example the MAC of his device). This key-identity pair is signed with the secret key of the certification authority K_C^{-1} and can be verified by both parties A and B even off-line.

Now given this setup, the secure-SP connection part extends the basic exchange of request and reply (`REQ`, `REP`) by additional time stamps T, nonces N (where indices $\in \{A, B\}$ indicate the sender and receiver), sender, and a symmetric session key K_S for the future data transmission. The contents of the following two messages are encrypted using the public keys K_A and K_B so that only the intended recipient A or B can read the message contents.

$$\begin{aligned} A \mapsto B &: \texttt{REQ} + \{\texttt{SYNC\_NO}_A, T_A, A, N_A\}_{K_B} \\ B \mapsto A &: \texttt{REP} + \{\texttt{SYNC\_NO}_B, T_B, B, N_B, N_A, K_S\}_{K_A} \end{aligned}$$

If this two step challenge response protocol succeeds, a connection between A and B is established. In the course of that connection, A and B can now exchange messages whose `SYNC_NO` and shared secrets N_B and T_A are cryptographically protected by the symmetric key K_S that has been exchanged.

$$A \mapsto B : \{\texttt{SYNC\_NO}_A, N_B, T_A, A\}_{K_S} + \text{data message}$$

Note, that the authentication of A to B is only complete after the third step, i.e. the first data transmission, where A shows possession of the private key K_A^{-1} by decrypting and re-encrypting N_B, T_A, and K_S. This protocol has been formalized and successfully verified with Avispa (for details see the following section). Confidentiality and integrity of the data communication part holds as long as the session keys are not broken. This additional assumption is necessary and explicit in Avispa: it is beyond the scope of the protocol verification since we abstract from key length and duration of use. The same applies for the above mentioned public key infrastructure.

The secured SP protocol's communication part bears a strong resemblance to the (corrected) Needham-Schroeder asymmetric authentication protocol. This is no surprise, as the NS-asymmetric protocol (NSPK) is the essence of remote authentication.

E. Formalizing and Verifying SP in Avispa

The SP protocol resembles a lot the improved version (not susceptible to man-in-the-middle attacks) of the NSPK protocol.

1. $A \rightarrow B : \{N_A, A\}_{K_B}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A}$
3. $A \rightarrow B : \{N_B\}_{K_B}$

Due to this resemblance and the fact that NSPK is available in the Avispa library, we can *reuse* in large parts the Avispa formalisation of that NSPK protocol as a starting point adapting it to SP.

The formalization has two roles alice and bob. We first consider alice.

```
role alice (A,B : agent,
    Ka,Kb : public_key,
            Ks: symmetric_key,
    Snd,Rcv : channel (dy)) played_by A def=
local
State : nat,
Na,Nb : text,
Ta,Tb : text,
REQ,REP : message,
SN,MN,AMN : text
init State := 0
transition
1. State = 0  ∧ Rcv(start) ⇒
   State':= 1 ∧ Na' := new()
              ∧ Snd(REQ.SN.Ta.A.Na'_Kb)
      ∧ witness(A,B,na,Na')
      ∧ secret(Na',na,A,B)
2. State = 1  ∧ Rcv(REP.SN.Tb.B.Nb'.Na'.Ks_Ka) ⇒
   State':= 2 ∧ Snd(SN.Nb.Ta.A_Ks)
              ∧ wrequest (A,B,nb,Nb')
end role
```

Alice's role specifies the protocol from here point of view, i.e., it comprises the initial sending step as a send, followed by waiting to receive the step two, and finally sending out the third step that already used a symmetric key for efficient transport level encryption. Complementing this the role for bob is as follows.

```
role bob (B,A : agent,
  Kb,Ka : public_key,  Ks: symmetric_key,
  Snd,Rcv : channel (dy)) played_by B def=
local
State : nat,
Na,Nb : text,
Ta,Tb : text,
REQ,REP : message,
SN,AMN   : text
init State := 0
transition
1. State = 0  ∧ Rcv(REQ.SN.Ta.A.Na'_Kb) ⇒
   State':= 1 ∧ Nb' := new()
              ∧ Snd(REP.SN.Tb.B.Nb'.Na'_Ka)
      ∧ witness(B,A,nb,Nb')
      ∧ secret(Nb',nb,A,B)
2. State = 1 ∧ Rcv(SN.Nb.Ta.A_Ks) ⇒
   State':= 2 ∧ wrequest(B,A,nb,Nb)
end role
```

Similar to before for the Radius protocol, the two roles are now instantiated into a session.

```
role session (A,B: agent,
      Ka, Kb : public_key, Ks : symmetric_key)
def=
local SA, RA, SB, RB: channel (dy)
composition
   alice(A,B,Ka,Kb,Ks,SA,RA)
∧ bob(B,A,Kb,Ka,Ks,SB,RB)
end role
```

Also an environment is defined that sets up the intruder.

```
role environment() def=
const ta,tb,sn,mn,amn : text,
a, b, i      : agent,
na, nb : protocol_id,
ka, kb, ki   : public_key,
        ks,ksi: symmetric_key
intruder_knowledge =
      {a,b,i,ka,kb,ki,inv(ki),ta,tb,mn,amn,sn}
composition
session(a,b,ka,kb,ks) ∧ session(a,i,ka,ki,ksi)
end role
```

The goal we check is as follows (equal to the guarantees in NSPK).

```
  secrecy_of na, nb
  authentication_on alice_bob_nb
  authentication_on bob_alice_na
```

The checking of this goal succeeds and provides a fully automated verification of the authentication process of our specified SP protocol. The engineering of this secure SP protocol has been a process of reuse (reusing the NSPK-Avispa specification in large parts) and also composition: the SP protocol – as such rather a transport protocol – has been composed into secure SP by prepending the above authentication steps. This is a composition process.

V. Conclusions

In this paper we have shown that an adapted version of the Radius protocol using SHA-256 instead of MD5 provides exactly the same security guarantees as the RFC version based on MD5. The verification is a fully automatic analysis in the Avispa toolkit, a specialized model checker for security protocols. We could generalize this result to guarantee security for Radius protocols using secure hash functions, even other than SHA-256. We furthermore illustrated on the example of the simple protocol SP that modelchecking can be used to stepwisely introduce security to a transport layer protocol. The verification process has shown the feasibility of model checking as an engineering tool.

Although the authors of [15] provide a model for the Radius protocol as defined in the RFC, they have failed to sufficiently generalize their results. In some sense, our approach resembles a *refactoring* of the formal model: refactoring is a technique from software engineering supporting the change in software without affecting desired properties; we change the formal model of Radius by replacing MD5 by SHA-256 *without losing desired security properties*. In the process of following the earlier design, we discovered that the model is by no means limited to the classical Radius but can indeed be generalized to a more secure Radius-SHA256, and that this generalization can be extended to arbitrary hashes.

The generalization or refactoring could be an interesting concept to explore because for the working security engineer it provides an easy to use extension making the rather complex model checking process easy to access and provide a practical tool to allow more flexibility in network security engineering. Apart from facilitating the process of protocol engineering, this could also advocate the use of formal specification and automated model checking in the domain of network security.

References

[1] Mahdi Aiash, Glenford Mapp, Aboubaker Lasebae and Raphael Phan. Exploring the Concept of Scope to Provide Better Security for Internet Services. *Proceedings at the First Global Conference on Communication, Science and Information Engineering.* Middlesex University, London, 2011.

[2] Avispa v1.1 User Manual. Available at http://www.avispa-project.org, 2006.

[3] D. Cottingham and P. Vidales. Is Latency the Real Enemy in Next Generation Networks? *Proceedings of First International Workshop on Convergence of Hetergeneous Wireless Networks*. July 2005.

[4] F. Kammüller, G. Mapp, S. Patel. A. S. Sani. Engineering Security Protocols with Modelchecking – Radius-SHA256 and Secured Simple Protocol. *International Conference on Internet Monitoring and Protection, ICIMP'12*, 2012.

[5] I.-G. Kim and J.-K. Choi. Formal Verification of PAP and EAP-MD5 Protocols in Wireless Networks: FDR Model Checking. *AINA*. 2004.

[6] Y. Kirsal-Ever. Development of Security Strategies using Kerberos in Wireless Networks. PhD Thesis, Middlesex University, 2011.

[7] G. Lowe. Breaking and Fixing the Needham-Schroeder Security Protocol. *Information Processing Letters*, 1995.

[8] G. Mapp, F. Shaikh, M. Aiash, R. Vanni, M. Augusto and E. Moreira. Exploring efficient imperative handover mechanisms for heterogeneous networks. *Proceedings of the International Symposium of Emerging Ubiquitous and Persuasive Systems*. Indianapolis, Ind, USA, August 2009.

[9] G. Mapp, F. Shaikh, J. Crowcroft, D. Cottingham, and J. Baliosian. Y-Comm: A Global Architecture for Heterogeneous Networking (Invited Paper). *3rd Annual International Wireless Internet Conference (WICON)*, 2007.

[10] yRFC2: The Simple Protocol (SP) Specification, 15.9.2012. http://www.mdx.ac.uk/research/areas/software/ycomm_research.aspx

[11] S. Patel. Implementation and Analysis of Radius Protocol using Avispa. Master's Thesis. Middlesex University, 2011.

[12] RFC 4960 - Stream Control Transmission Protocol. *IETF.* September 2007.

[13] R. Rivest. Message-Digest MD5. *Network Working Group, RFC: 1321*. http://www.kleinschmidt.com/edi/md5.htm. 1992.

[14] A. S. Sani. Verifying the Secured Simple Protocol in AVISPA. Master's Thesis, Middlesex University, 2012.

[15] V. Sankhla. Formalisation of Radius in Avispa. http://www.avispa-project.org/library/RADIUS-RFC2865, University of Southern California, 2004.

[16] Rogue CA certificate signed by a commercial Certification Authority. http://www.win.tue.nl/hashclash/rogue-ca/\#sec71 Presented at the 25th Chaos Communication Congress, Berlin 2008.

[17] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. *Advances in Cryptology, Eurocrypt 2005.* LNCS **3439**, Springer 2005.

[18] Wikipedia RADIUS, http://en.wikipedia.org/wiki/RADIUS, 2012.

[19] Remote Authentication Dial In User Service (RADIUS). RFC 2868. Internet Engineering Task Force (IETF). http://tools.ietf.org/html/rfc2865, accessed 18th December 2012.

[20] Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes. RFC 5080. Internet Engineering Task Force (IETF). http://tools.ietf.org/html/rfc5080, accessed 18th December 2012.

[21] The Ycomm Framework. Offical Web-Site, Middlesex University. http://www.mdx.ac.uk/research/areas/software/ycomm_research.aspx.