Institut Eurécom
Department of Mobile Communications
2229, route des Crêtes
B.P. 193
06904 Sophia-Antipolis
FRANCE

Research Report RR-09-235

Computer Aided Design of a Firmware Flashing Protocol for Vehicular On-Board Networks

September 28th, 2009

Muhammad Sabir Idrees, Yves Roudier

Tel : (+33) 4 93 00 81 90
Fax : (+33) 4 93 00 82 00
Email : {Muhammad-sabir.idrees, Yves.roudier}@eurecom.fr

[1]Institut Eurécom's research is partially supported by its industrial members: BMW Group Research & Technology - BMW Group Company, Bouygues Télécom, Cisco Systems, France Télécom, Hitachi Europe, SFR, Sharp, STMicroelectronics, Swisscom, Thales.

Computer Aided Design of a Firmware Flashing Protocol for Vehicular On-Board Networks

Muhammad Sabir Idrees, Yves Roudier

Abstract

Vehicular On-Board Networks consist of up to 70 electronic control units (ECUs) interconnected by buses and gateways and organized within domains with different trust levels. This paper describes how the design of a protocol for deploying a new firmware onto various vehicular ECUs might be automated. In particular, such a protocol should prevent attacks to the firmware update process and make sure that no malicious firmware is actually installed in place of a regular firmware update, despite the fact that it may be sent through insecure domains. Designing security protocols for ECU communication in such architectures can become quite complex and error-prone, especially given the computational and deployment constraints that apply in the domain. This paper discusses how the protocol designer might receive some help in exploring fundamental design decisions based on the systematic review of alternative security architectures and potential threats.

Index Terms

Vehicular on-Board Network, Protocol Design, Security Properties

Contents

List of Figures

1 Introduction

Future visions of road transportation include networked vehicles and intelligent transport systems (ITS) that will enhance the safety of drivers and other road users. The on-board network of these vehicles contains more than 70 *electronic control units (ECUs), electronic sensor* and *electronic actuators* interconnected by buses and organized in different domains. Depending on the communication task and system requirements such as flexibility, modularity, scalability, and fault detection properties, each domain control unit define separate communication network topology and use different Bus system i.e. LIN, CAN, FlexRay or MOST as shown in Figure 3.

Numerous firmwares are installed in the ECUs to enable various functionalities, for instant vehicle control or maneuverability. In order to verify the state of the vehicle (system functionality, security), regular firmware diagnosis and updates are required. As of now, the firmware diagnosis and update process is done off-board, by connecting (hardwired) a diagnosis tool with the on-board network and performing firmware updates. However, in the future, it will be possible to perform remote diagnosis and the remote flashing. This will provide several advantages over hardwired access, such as faster firmware updates, time saving and it will improve the efficiency of the system by installing firmware updates as soon as they are released by the car manufactures.

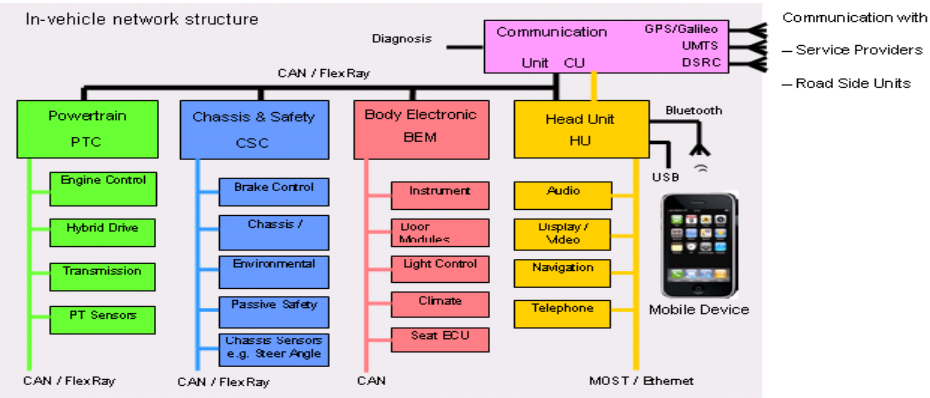

Figure 1: On-Board Reference Architecture [1]

The remote firmware flashing [1] process relies on a diagnosis request remotely sent by a service station to the Communication Unit (CU) of an on-board network. A possible consequence of diagnosis would be the update of firmware issued by the car manufactures, to improve the functionality of the system. A diagnosis tool requests the information it needs from the ECU (ECU type, firmware version, etc) in order to update the firmware.

Most existing solutions for implementing firmware update protocols rely on the development of the lightweight security protocols over the air (SFOTA) [3] and developing a framework for self-verification of the firmware update [4], in order to provide end-to-end security. Although these solutions provide a secure firmware update on a wireless link moreover, the verification of the correct firmware by using virtualization techniques. These solution does not consider the impact of firmware updates on the whole system architecture of the vehicle, e.g., in [3], symmetric encryption technique is used and it does not deal with key management issues. Several assumptions regarding trust relationship between vehicle and portal[1] are made, such as that the portal always sends the correct firmware. Furthermore these approaches do not consider the intrusion and denial of service attacks on vehicle and portal. However, there are numerous scenarios, where an attacker attacks one or both, the portal (service station in our use case) system and the vehicle's on-board network, in order to harm driver, gaining information about the driver, financial gains, gain information about vehicle manufacturer's technology or to gain personal advantages. Several attacks are identified in [2] for on-board vehicular network, i.e., gaining access to other domains of the on-board network, the service station injects bogus authority keys into the ECU which compromise the overall security of vehicular on-board architecture.

The design of security protocols for such an architecture can become quite complex and error-prone, especially when information flows from different trusted to untrusted domains and vice versa. Our solution to these problem relies on the development of an expert system for firmware flashing protocols. It is still under development, but it has been able to guide the system architecture designer to consider possible design solutions to implement firmware flashing protocol. These design solutions will prevent the development of inconsistent security solutions and eventually reduce the computation and deployment constraints.

The remainder of this paper is structured as follows. Section II discusses the design methodology. In section III, discusses the design space specification. Section IV describes related work. Section V presents the future work and Section VI concludes the paper.

2 Design Methodology

The design of security system usually requires a fully specified system architecture and it should consider all possible critical situations, indeed, ignoring even single security factor can lead toward inconsistent security solutions. In order to consider critical situations during firmware flashing process, we perform an attacker model and threat analysis. Based on attacks and threat analysis, we specify several security requirements to secure firmware flashing and to ensure overall system security.

[1]The portal is the central unit surrounded by the vehicles, communicates with the vehicles over the wireless connection.

2.1 Attacker Model

There are two possible ways to attack vehicular on-board network, through firmware flashing process [2].

- Attacker abuse the flashing itself in a service station (workshop).
- Gaining access to the Communication Unit (CU) through different resources, i.e., internet access or personalizing the car.

An attack tree [5] approach is used to capture possible attacks against vehicular on-board network. Some scenarios of attacks could be that an attacker abuses the flashing itself in a service station and installs modified firmware into on-board ECU. On the vehicle side, even if all security checks (authenticity, integrity, confidentiality, etc) are performed, the attacker can get access to other domains of the on-board network. The attacker can also get access to the Communication Unit by exploiting vulnerabilities in protocol implementation. If it is not the case, it is still possible to attack Communication Unit via the Head Unit through internet access or personalizing the car with external devices such as PDA, Laptop, etc, to abort the firmware flashing. Another possible attack could be injecting bogus authority keys into ECUs, which allows the attacker to send fake messages to other domains within the on-board network or broadcasting the fake warning messages to other vehicles and Roadside Units (RSU). More detailed attacks on vehicle on-board network, during firmware flashing process, are shown in Figure 2.

2.2 Threat Analysis

Threat analysis is performed in order to identify the most significant assets under attack and the level of risk posed by potential attacks on the vehicular on-board network. The level of risk is defined by the severity of the attack and the probability of an attack to be successfully performed. As mentioned in [2], severity of an attack is measured with respect to operational, safety, privacy and financial aspects whereas probability of success of an attack depends on the attack potential such as expertise, knowledge about the vehicle on-board system and the time required to perform the attack. For example, in order to attack the firmware flashing process, the attacker should be an expert, that has knowledge about the underlying algorithms, protocols, hardware structure, security behavior, principle and concepts of security employed in vehicle on-board network. Depending on the attack potential (basic, enhanced basic, moderate, high or beyond high) an attacker can attack on-board assets such as Powertrain Controller, Powertrain peripheral, Head Unit, In-car Sensors, Chassis&Safety controller and the Communication Unit. The purpose of threat analysis is to determine and classify the attack potential for each attack identified in the Figure 2, and to give threat analysis as an input to the expert system.

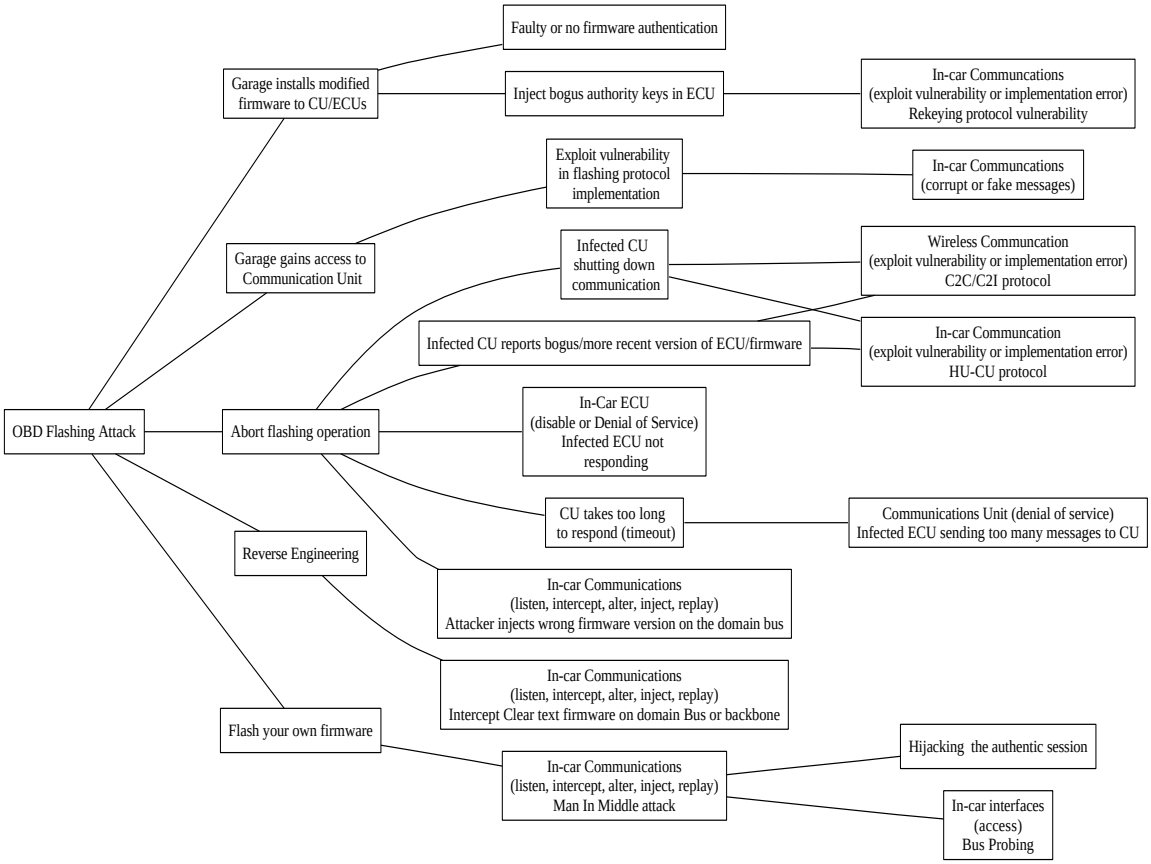

Figure 2: Attacks on Firmware Flashing Process [2]

2.3 Security Requirements

An analysis of attack tree shows that specific attacks may contribute to different attack objective and into different attacks. Several security requirements are defined to perform secure firmware flashing and to protect overall system against attacks. These security requirements are the following:

- *Code Origin Authenticity*: This requirement will ensure that, whenever a flashing command is sent to ECU, the origin of code is ensured. The receiver can verify that the received firmware is really from car manufacturer. This requirement will prevent attacks from installing faulty firmwares.
- *Message Source Authenticity*: This security requirement will ensure that, whenever a message arrives at vehicle reception, i.e., the Communication Unit, message source is authentic.
- *Code Integrity*: This security requirement will ensure that the firmware data, received as an update, has not been modified since it left the manufacturer

servers. This requirement will prevent attacks from installing faulty or modified firmware into the ECU.

- *Flashing Command Integrity*: Whenever ECU receives flashing command request, the integrity of flashing command must be ensured in order to prevent attacks against sending fake commands for firmware flashing.
- *Firmware Data Confidentiality*: This requirement will ensure that the firmware data should remain confidential, when updates are distributed by the manufacturer.
- *Firmware Update Confidentiality*: This requirement will ensure that the attacker should not gain information out of the flashing process about the version of firmware being installed or the ECU being updated.
- *ECU/CU Availability* : This requirement will ensure that the CPU, RAM, Bus of ECU/CU are available throughout the flashing process.
- *Flashing Command Freshness*: This requirement will ensure that all the command sent to the ECU for firmware flashing, possess freshness property.

3 Specifying the Design Space

Analyzing the design of an embedded system like a vehicular on-board network requires combining multiple information about its architecture. This includes the description of the individual domains and bus systems, the network topology defined in each domain, the ECUs available, information flow among different domains and information about external devices (e.g., mobile phones, PDA, laptop, Diagnosis Tool), etc. whose interconnection is forming the on-board network. This collection of information will form the basis for the expertise driven process of specifying security solutions.

3.1 System Representation

The Prolog language [6] introduces a fairly declarative way of describing and matching logical patterns[2] and has long been used both as a prototyping tool and as a way to verify logical properties. We decided to introduce descriptions about the architecture of vehicular systems and the assumptions made at design time using this language for both objectives, as well as an expert system for introducing security-related design principles. All such information forms a knowledge base as presented in the listings below. Prolog facts are used to state features and properties that are unconditionally true for on-board networks. For instance, `extCom('DT','CU')` states that CU communicates with an external entity DT. Different security properties can also be specified using Prolog fact expression.

[2]Prolog also can handle numerical constraints, which we don't use in this paper

For instance, different key sizes might be specified about encryption algorithms, depending on their availability or security. The Prolog program and interpreter allows the architecture designer to query the expert system in order to obtain alternative design solutions automatically, based on Listing 1 .

```
% Communication Channels
extCom('DT','CU').
intCom('CU' ,'PTC').
domCom('PTC','CSC').
busCom('PTC','ENG').
truCom('ECU1','ECU2').

% Network Topology
ptcBusSystem('CAN','LINE').
huBusSystem('MOST','Star').
cscbusSystem('CAN','LINE').

% ECU Properties
ecuType('MPC555','32Bit','26K','448K').
ecuType('MPC565','32Bit','64K','1.5M').

% Encryption Algorithms
symCrypt('3DES','48','168','64').
symCrypt('AES','1032','256','128').
asymCrypt('RSA','2048','123K').
asymCrypt('RSA','1024','123K').
```

Listing 1: Facts and Rules about On-Board Network Architecture

3.2 Introducing the Security Expertise

The aim of the Prolog program is to allow the architecture designer to query the expert system in order to obtain alternative design solutions automatically.

Queries can for instance be answered by combining the inference rule `flash(A, M, B, E)` and free variables using the unification mechanism of Prolog. A query is based on one or more goals. Executing the expert system program results in the exploration of the knowledge base using the Prolog engine and its backward chaining inference mechanism. This approach makes it possible to determine if precise properties are true or false, after all the requirements and constraints defined about the architectures have been logically combined together. This mechanism is of specific interest for us in that it derives all the system designs that are valid under all the conditions stated in the knowledge base. For instance, the tool can search for all situations that match with the assurance that all messages sent from *A* to *B* are confidential, based on facts, which are assumed to be true or given, and inference rules, which describe security best practices, technical or financial constraints, deployment and real-time constraints, etc. In case the ECU *A* com-

municates in cleartext with ECU *B*, the program execution would fail and return a negative answer.

Using Prolog as the basis for our expert system also makes it possible to evaluate queries with free variables. Prolog introduces a unique pattern-matching algorithm, termed unification, to explore solutions of such only partially explicit logical statements. For instance the program might walk through goals and search for all possible authorized messages between ECU *B* and ECU *E*.

In our example, the resolution of the *flash* clause starts with the evaluation of its first statement `extCom(A,M,B)` which proves that a specified message *M* is sent by the sender ECU *A* to the receiver ECU *B*. The unification algorithm work its way through the statements until it succeeds in unifying its goal with the head rule or with a fact. If the goal cannot be unified with the facts the tool will respond negative. The inference rules allow us to establish a multistep message path from *A* to *E*.

```
% Inference Rules
flash(A, M, B, E):- extCom(A,M,B),
    intCom(B,M,X), busCom(X,M,E),
authenticity(A,M,B),
integrity(A,M,B),
freshness(A,M,B).
```

Listing 2: Security Related Inference Rules

The program in this example cares for two different concerns. The first one deals with communication over the on-board network and aims at establishing the data flow between A and E and at finding out what are the properties of the different entities involved in the implementation of a distributed function. The Prolog program can be seen both as a specification of the system and as its simulation. Thanks to this, threat analysis might be introduced at this stage based on the unfolding of a scenario, although our example only describes a valid system or valid security practices based on constructive properties. e detailed multistep message path approach allows the expert system to consider threat analysis and identified security requirements for each entity involved in firmware flashing process.

The architecture designer can query the tool based on facts that describe its integrity `integrity(Sender, Message, Receiver)`, to ensure the code integrity requirement for a distributed function. It can also query the tool in order to find the best possible place for implementing firewalls. For instance, this might help solve questions like: can a filtering firewall be implemented at the domain gateway level to enforce access control on the basis of the packet header or should this be enforced through end-to-end authentication with the ECU only?

3.3 Discussion

The use of Prolog provides a design exploration mechanism almost for free but our experience with the demonstrator shows some areas for immediate improve-

ment which we are working on. In particular, we plan to buffer results that were proven in other analyses, which a completely declarative Prolog program does not achieve; while this is not so much of a problem with purely logical rules, we expect to integrate numerical constraints to our analysis. Enumerating all possible solutions may also be quite time consuming and we are currently working into introducing heuristics to eliminate unsuccessful designs as soon as possible. Both issues mean that the expert system rules cannot be as declarative as one would hope. In particular, automotive industry engineers should likely contribute to defining their organization in order to fine tune the general purpose resolution mechanism of Prolog with their strategies derived from their experience.

4 Firmware Flashing Protocol Design

The flashing protocol design specifies how the two entities, service station and the vehicle can, securely communicate with each other and perform firmware updates without compromising the overall system security, i.e., it specifies the messages that can be interchanged and the design solutions that should be considered during a firmware update. A comprehensive functional view for firmware flashing process is shown in figure 3, where different components exchange messages among each other. These components are running on different on-board units.

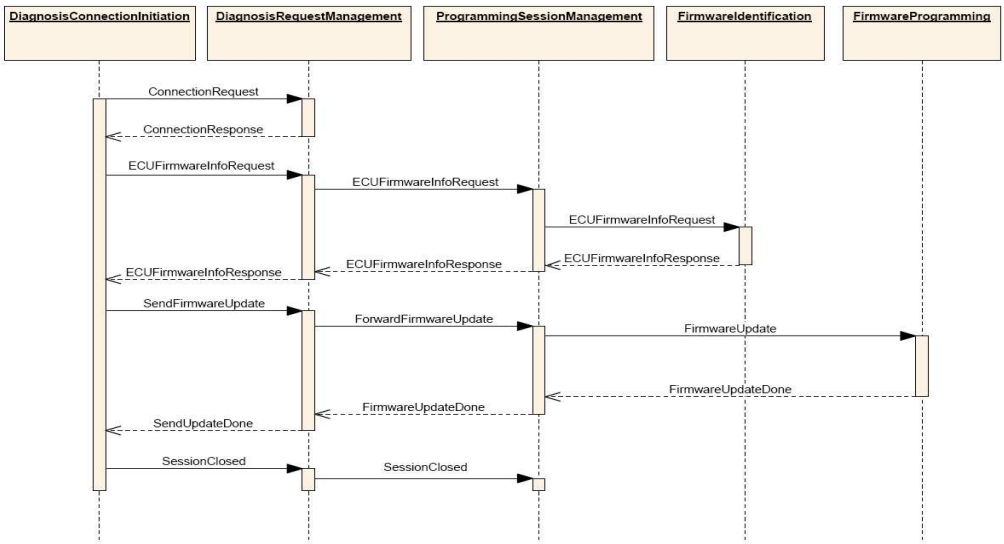

Figure 3: Functional View of Firmware Flashing Process

The `DiagnosisConnectionInitiation` component, running on the diagnosis interface unit, sends a connection request to `DiagnosisRequestManagement` component. A connection answer is then sent and session keys are shared to allow secure communication channel. To know which version is running on the ECU, a `ECUFirmwareinfoRequest` message is sent to `FirmwareIdentification`

component. If the type is the expected one then the flashing session started. The flashing tool sends a request to open a programming session to *ProgrammingSessionManagement* component. Once the programming session is started the flashing tool sends the encrypted new software to the *FirmwareProgramming* component. At the end, the flashing tool closes the programming session at ECU level (*ProgrammingSessionManagement*).

Based on the functional view model, the architecture designer queries the tool for possible design solutions as shown in listing 3, which states that, the sender *DT* sent a request to *CU* to update ECU firmware in the PTC domain.

```
% Query
?- flash(DT, "Firmware Update Message", CU, ECU–PTCDomain).
```

Listing 3: Query for Firmware Flashing Design

Executing the query defined in listing 3, program results in generation of multicast message path: $DT \rightarrow CU \rightarrow PTC \rightarrow ECU$, based on the inference rules defined in listing 2. While establishing the path, the expert system also caches the characteristics of each entity involved in this firmware update process. This allows the generation of firmware protocol design based on the computation and deployment constraint of each entity. For instant, the design solution for firmware authenticity is based on listing 4.

```
authenticity(A, M, E):-
  ecuTypeCheck(ECU, E), chartEvu(Crypt, ECU).
```

Listing 4: Inference rule for Authenticity

In this rule the authenticity clause starts with the evaluation of its first statement *ecuTypeCheck(ECU, E)*, which deduce the properties of received ECU then the program walk through the next rule and evaluates the suitable cryptographic algorithm. For example the requested ECU is MPC555, the program caches all the property of that ECU and move to the next rule and evaluate the appropriate cryptographic algorithm for MPC555. Based on the ECU characteristics, it returns the RSA encryption algorithm with 2048-bit key size. The program returns back to the main goal, Listing 2 . In the same way the program applies other inference rules (integrity, freshness, etc) on the firmware flashing process. In the end of program it returns both analysis of each rule and appropriate design solution to the architecture designer.

5 Related work

5.1 Design of Firmware Update Protocols

A security architecture for secure software upload in vehicular network via wireless communication link is presented in [7]. The proposed security architecture is based on the authentication key mechanisms, where keys are used to establish secure link using SSL, VPN or any secure mode. After establishing a session,

symmetric encryption keys are exchanged to send the software in an encrypted form. Another solution proposed in [7] is to send multiple copies of software in order to improve the security level of the transmitted data. After some random time interval the supplier again establishes the link with the vehicle and sends a second copy of the software, the vehicle compares the two copies sent by the supplier and, if the two copies are same, vehicle sends an acknowledgment to supplier and automotive company, otherwise it asks to retransmit unmatched packets. This approach imposed several system constraints in order to ensure the secure software upload such as memory size, bandwidth and length of encryption keys. In [3] a lightweight protocol for secure firmware updates over the air (SFOTA) in intelligent vehicles is proposed. In SFOTA protocol, data integrity can be achieved by forming a hash chain, each transmitted fragment is hashed and included with the previous fragment. Verifying the signature of the first hash, provides the data authentication. In order to provide the data confidentiality, symmetric encryption (CBC mode) is used. Furthermore the data freshness property can be also achieved by receiving the signed packet, since each packet contains a hash of the previous packet. The vehicle can verify the order of received packet to attain data freshness. In [4] a framework for self-verification of firmware update over the air in vehicle ECUs is proposed. In their work a verification code concept is included in the transmission, integrity of firmware update can be assured by verifying the verification code. In the verification protocol, service portal generate the random challenge, calculates a hash chain of the firmware, and forwards verification code, firmware binary and the challenge to the vehicle using [3] SFOTA protocol. In the vehicle, these challenges and verification codes are stored in the control system. They are accessed by ECU using Virtulization techniques.

5.2 Prolog Based Protocol Verifiers

A Prolog based automatic cryptographic protocol verifier is proposed in [8]. A simple intermediate representation of protocol is developed using prolog rules and facts. In their work, an attacker concept is introduced to prove the secrecy properties, such as determining whether the attacker can get the secret or not. The key advantage of their verifier is that the algorithm does not limit the number of runs and, if the verifier does not find the flaw, then there is no flaw in the protocol which provides the real security guarantees.
In [9], a model of computation for the Naval Research Laboratory (NRL) Protocol Analyzer is presented. The main intent of this protocol analyzer is to guide the user in proving the cryptographic protocol. A set of prolog rules and facts are used to define the protocol. The protocol analyzer analyzes the insecure states and give the detail description of all possible states. Furthermore the NRL protocol analyzer allows the user to check the reachability property. In [10], a prolog based tool: *The Interrogator* is explained. The interrogator search the security vulnerabilities in cryptographic key distribution protocol. The protocol and all other assumptions are specified in prolog program. During program execution, the program finds the

traces of message modification in the protocol and alerts user about attacks. However, these approaches provides the post analysis and verification of implemented cryptographic protocols, using Prolog profile. In contrast, our approach is based on pre-analysis of the system architecture and aims at suggesting fundamental design solutions for implementing security protocol.

6 Conclusion and Future Work

In this paper we discussed the interest of using Prolog as an expert system for developing a firmware flashing protocol and presented the organization of our current prototype. The objective of this approach is to assist the protocol designers in reviewing fundamental design decisions. In particular, automation makes it possible to evaluate the most complex combinations of the system architecture, its potential threats, security requirements, and deployment constraints (computation, cost, performance), thereby offloading the designer from his most cumbersome tasks. We are currently working on enhancing the performance of our expert system implementation.

7 Acknowledgments

This work has been carried out in the EVITA (E-safety vehicle intrusion protected applications) project, funded by the European Commission within the Seventh Framework Programme (FP7), for research and technological development.

References

[1] E. Kelling, M. Friedewald, M. Menzel, H. Seudie, and B. Weyl, "EVITA - D: 2.1 specification and evaluation of e-security relevant use cases," Tech. Rep., Feb, 2009. [Online]. Available: http://evita-project.org/Deliverables/EVITAD2.1v1.1.pdf

[2] A. Ruddle, Y. Roudier, S. Idrees, B. Weyl, M. Friedewald, T. Leimbach, A. Fuchs, S. Grgens, O. Henniger, R. Rieke, M. Ritscher, H. Broberg, L. Apvrille, R. Pacale, and G. Pedroza, "EVITA - D: 2.3 security requirements for automotive on-board networks based on dark-side scenarios," Tech. Rep., March, 2009. [Online]. Available: http://evita-project.org/Deliverables/EVITAD2.3.pdf

[3] D. Nilsson and U. Larson, "Secure firmware updates over the air in intelligent vehicles," in *Proc. IEEE International Conference on Communications Workshops ICC Workshops '08*, 2008, pp. 380–384.

[4] D. Nilsson, L. Sun, and T. Nakajima, “A framework for self-verification of firmware updates over the air in vehicle ECUs,” in *Proc. IEEE GLOBECOM Workshops*, 2008, pp. 1–5.

[5] B. Schneier, “Attack trees,” *Dr. Dobb’s Journal*, 1999. [Online]. Available: http://www.schneier.com/paper-attacktrees-ddj-ft.html

[6] A. Colmerauer and P. Roussel, “The birth of prolog,” *History of Programming Languages, ACM Press / Addison-Wesley*, 1996.

[7] S. Mahmud, S. Shanker, and I. Hossain, “Secure software upload in an intelligent vehicle via wireless communication links,” in *Proc. IEEE Intelligent Vehicles Symposium*, 2005, pp. 588–593.

[8] B. Blanchet, I. Rocquencourt, and L. C. Cedex, “An efficient cryptographic protocol verifier based on prolog rules,” in *In 14th IEEE Computer Security Foundations Workshop (CSFW-14*. IEEE Computer Society Press, 2001, pp. 82–96.

[9] C. Meadows, “A model of computation for the NRL Protocol Analyzer,” in *Proc. Computer Security Foundations Workshop VII CSFW 7*, 1994, pp. 84–89.

[10] J. Millen, S. Clark, and S. Freedman, “The interrogator: Protocol secuity analysis,” vol. SE-13, no. 2, pp. 274–288, 1987.