Formal Verification of Payment protocol using AVISPA

Rizwana Shaikh A.R. and Satish Devane
SIES Graduate School of Technology
Ramrao Adik Institute of Technology
India

Abstract

Emerging e-commerce activity is giving scope for the design of many new protocols, and to gain confidence, these protocol need to be verified for its designed properties. Specifically protocol used in e-commerce transactions needs to be verified for their security properties. Verification of these protocols is done using the formal verification tools. AVISPA is one of the evolving tools used mainly for verifying security properties. A newly designed electronic payment protocol is verified for its correctness and security properties. This paper discusses various formal verification methods and tools. Also it presents the use of AVISPA for verifying the security properties of the newly evolved electronic transaction protocol.

1. Introduction

This era of Internet has seen increasing use of smarter electronic devices to ease life. Specifically in electronic payment system, use of smart card is making its own position for online transactions. To gain customer confidence, protocols used in such payment system should be robust and secure. Lots of such protocols have been already in use [7], but many newly emerging payment protocols are there which needs to be verified for security and correctness of properties. Also the properties of specification with respect to service provided by the protocol, security properties and any other additional property used to indicate that the protocol is foolproof.

Verification of the protocol is a major activity in their development to get the user confidence. Formal verification is one of the methods used for verification. Formal Methods increases understanding of systems, increases clarity of description and helps to solve problems and remove errors [10]. Use of Formal Methods increases dependability and usability of systems. Many tools are available for verification of the protocols. These tools like Murphi [4], CSP [2], FDR [5], NRL protocol analyzer [3], Isabelle [6] are already available and in use. Also many advanced new tools are evolving, one of which is AVISPA [1].

AVISPA is an evolving tool, only to verify the cryptographic properties of the protocol. This tool is developed by the institutions like Artificial Intelligence Laboratory, DIST, University of Genova, Italy and Siemens Aktiengesellschaft, Munich, Germany. The objective is to develop a rich specification language for formalizing protocols, security goals, and threat models of industrial complexity and tune this tool and demonstrate proof-of-concept on a large collection of practically relevant, industrial protocols.

AVISPA is a push-button tool for the automated validation of Internet security-sensitive protocols and applications [1]. It provides a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of automatic analysis techniques.

Various protocols have been verified using available tool in past. The protocol used in payment system needs to be verified for its correctness and for its security properties. The tool adapted is AVISPA. The newly designed protocol used in the smart card based payment system is outcome of the PhD Thesis of [9].

2. Formal Method Process

Formal Method is the process of verifying system behavior using formal semantics. The process of Formal method shown in Figure below: consists of the following three tasks:

1. Formal Specification – descriptive
2. Formal Synthesis-layered approach
3. Formal Verification – analysis

Formal specification abstracts the system by providing high level mathematical model of the system. It is used for high-level design and to catch ambiguous or inconsistent specifications. It can be; Model-oriented, Property-oriented, logical and "what" over "how", etc.

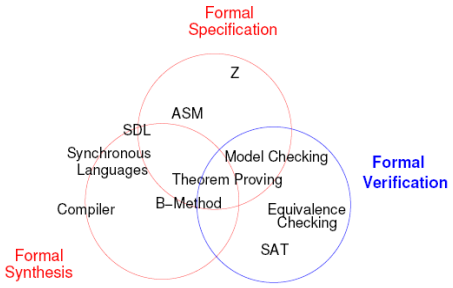

Figure1. Formal Method

Formal synthesis is a top-down and incremental refinement process which splits large verification tasks into smaller one. Refinement works but is costly, since each refinement step uses formal verification methods and more power full verification algorithms allow more automation.

Formal Verification is the process in which specification and system are given. Formal verification checks that system fulfills specification. It provides least change in development process. Actually a full complete verification is really difficult. So it requires simplifications and focuses on simple partial specifications for example, type safety, functional equivalence of two systems etc. Various methods are available which are implemented in tools. There exist simple algorithms for deducing properties directly and complex algorithms for hard or even un-decidable problems.

The above Figure shows various methods that provides the three tasks of Formal Methods. Some methods provide only synthesis, some provide only synthesis and some provide only verification. The intersection in the diagram shows that some methods provide all the three processes of Formal Verification.

2.1 Formal Verification and Simulation

Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output. Simulation embodies the principle of "learning by doing" i.e. to learn about the system, we must first build a model of some sort and then operate the model. Within the overall task of simulation, there are three primary sub-fields: model design, model execution and model analysis. Simulation is incomplete [11]. It needs to generate expected behavior. It is difficult to cover corner cases in simulation. It is CPU intensive. Formal verification is complete with regard to a specification. It does not need to generate expected behavior. Corner cases are automatically taken care of in formal verification. Magellan's unique hybrid architecture combines the strengths of new, advanced formal engines with the strengths of a built-in simulation engine to verify properties on large and complex designs.

2.2 Formal Verification Methods and tools

Formal method is the fancy name given to good old mathematical modeling, as applied to computer systems specification and design. When used well, a formal method should yield specifications that have significant advantages over natural language documents: they are unambiguous, they can be checked mechanically, and they are open to proof. So formal verification can be used as the process of testing [12].Some of the formal methods are Abstract State Machines, B method, Theorem proving, Model checking etc. Some of the formal methods are discussed as follows:

Abstract State Machines

In ASM, any algorithm can be modeled at its natural abstraction level by an appropriate ASM [13]. It consists of a methodology based upon mathematics which would allow algorithms to be modeled naturally. The ASM methodology has the desirable characteristics of Precision, faithfulness, understandability, executability, scalability and generality. There are lots of tools available to implement ASM like ASM Gofer and ATGT.

B-Method

The B-Method is designed to provide a notation and a method for requirement modeling, software interface specification, software design, implementation and maintenance, thus supporting the major phases of a software process [14]. The B-Method uses a simple 'pseudo' programming language to model requirements, to specify interfaces, and to provide implementations and intermediate designs. The language is known as AMN (The Abstract

Machine Notation). B-method provides lots of tools for the software development life cycle.

Theorem Proving

Theorem proving as defined in [15] is a technique where both the system and its desired properties are expressed as formulas in some mathematical logic. This logic is given by a formal system, which defines a set of axioms and asset of inference rules. Theorem proving is the process of finding a proof of a property from the axioms of the system. Steps in the proofs make use of these axioms and rules.

Model Checking

Modal checking is a technique that relies on building a finite model of a system and checking that a desired property holds in the model or not. As specified in [15], Model checking has been used primarily in hardware and protocol verification. Currently it employed in software system also. Two approaches for model checking are used. First is temporal model checking in which specifications are expressed in a temporal logic and systems are modeled as finite transition system. In the second approach the specification is given as an automaton and the system is also modeled as automaton, and is compared to the specification to determine whether its behavior conforms to the specification.

Method Integration

Two or more methods can be combined to provide effective verification. For example, one of the commonly method integration is Model checking and theorem proving given in [15]. It is the most promising directions in method integration, ideally to benefit from the advantages of both approaches.
These formal methods are implemented in terms of tools and used for verification. Some of the tools are discussed as;

Murphi

Murφ is a general purpose enumeration tool, can be used to analyzed cryptographic protocols. It is a protocol verification tool that has been successfully applied to several industrial protocols, especially in the domain of multiprocessor cache coherence protocols and multiprocessor memory models [4].

To use Murφ for verification, one has to model the protocol in the Murφ language and augment this model with a specification of desired properties. The Murφ system automatically checks, by explicit state enumeration, if all reachable sates of the modal satisfy the given specification. State enumeration can be done either using breadth first or depth first search. It has been used for several different verification tasks like verification of the cache coherence protocol; link-level and so on.

CSP (Communicating Sequential Process)

CSP is a language which allows the modeling of the communication of an inherently asynchronous composition of protocol sessions [2]. Each instance of an agent trying to execute the protocol is modeled by a CSP process that alternates between waiting for a message and sending a message (replying). Channels are used for communication between processes (between participants in the protocol).

FDR (Failure Divergence Refinement)

FDR is a model checking tool used along with the CSP [16]. Protocols can be brought in to the framework of CSP and potentially verified using the FDR model checker. In FDR the user had to provide a description of the adversary. With the development of Casper, which is a front-end to FDR, the construction of the adversary is automated. FDR has been successfully used to analyze security protocols.

NRL Protocol Analyzer

The NRL Protocol Analyzer, developed by the US Naval Research Laboratory, is a specialized tool for analyzing security protocols [3]. The NRL protocol analyzer is prototype special-purpose verification tool, written in prolog, which has been developed for the analysis of cryptographic protocols. NRL protocol analyzer can be used to prove security properties of cryptographic protocols as well as locate security flaws.

WSAT (Web Service Analysis Tool)

WSAT is a tool for analyzing and verifying composite web service designs, with the state of the art model checking techniques [17]. Web services are loosely coupled distributed systems communicating via XML messages. Communication among web services is asynchronous, and it is supported by

messaging platforms such as JMS which provide FIFO queues to store incoming messages. Data transmission among web services is standardized via XML, and the specification of web service itself (invocation interface and behavior signature) relies on a stack of XML based standards (e.g. WSDL). The final model generated is verified using SPIN.

Isabelle

Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus [6]. The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols.

Compared with similar tools, Isabelle's distinguishing feature is its flexibility. Most proof assistants are built around a single formal calculus, typically higher-order logic. Isabelle has the capacity to accept a variety of formal calculi. The distributed version supports higher-order logic but also axiomatic set theory and several other formalisms.

SPIN

SPIN is a popular open-source software tool that can be used for the formal verification of distributed software systems. SPIN was developed at Bell Labs in the original UNIX group of the Computing Sciences Research Center, starting in 1980 [18]. The software has been available freely since 1991, and continues to evolve to keep pace with new developments in the field. In April 2002 the tool was awarded the prestigious System Software Award for 2001 by the ACM.

AVISPA

AVISPA is a push-button tool for the automated validation of Internet security-sensitive protocols and applications [1]. It provides a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of automatic analysis techniques.

Our focus here is on AVISPA for verifying security properties of electronic payment protocol which is discussed in the next section.

3. Electronic transaction protocol

A protocol is a set of rules and conventions that define the communication framework between two or more agents. These agents, known as principals, can be end-users, processes or computing systems. A protocol can be of any type like communication protocol, cryptographic protocol, network protocol or payment protocol etc. A protocol should satisfy some characteristics. It has to be tested for its simplicity, robustness, modularity and consistency. If the protocol is for payment transactions, the completeness of the protocol depends upon the transactions finally getting settled at back-end system. Payment transactions of vendors, whose salary are on daily basis, are of small value and more in numbers. Design and Implementation of e-purse scheme should be cost effective for the scheme provider and should facilitate migration from paper money to e-money. Thus, there is a need to have an affordable batch transaction transfer scheme which should work in online mode using some facility having a PSAM (Personal Secure Authentication Module) terminal with modem, or in offline mode using some novel approach.

Issues pertaining to online/offline transfer are addressed in the new smart card based payment transaction protocol. It is discussed in the following subsections.

3.1. What is Transaction Protocol ?

A transaction is a set of operation to perform single logical work. Many protocols are available to provide transaction between two or more parties. Formal techniques of specification and verification are used better to verify electronic payment protocols. The application of formal techniques to the modeling and design of electronic commerce protocols should help to improve their reliability. The electronic payment protocol described in this report mainly deals with small set of transaction that is transferred in terms of batch. The functional design and properties of the protocols gives detailed description of the protocol.

3.2. Functional Design of Protocol

The protocol is designed in two different ways. One way is between Merchant (M) and Merchant Acquirer (MA) called D_TTBP (Direct Transfer of Transaction Batch). The other way is via Mobile Merchant Acquirer (MMA). MMA is an authorized agent of MA. He (MMA) will move from merchant to merchant, collecting merchant's transaction batches on his own card MMAPSAM, which is a special

acquirer's floating card personalized by PSAM creator. These collected transaction batches by this MMA will finally be downloaded to MA for settlement either from a centralized location or by connecting MA directly from his terminal or by logging to the MA's internet site from a local internet terminal with smart card reader attached to it.

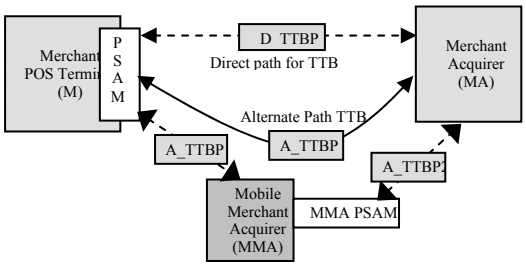

Figure 2 Transaction flow of the protocol

Figure 2 shows the transaction flow between Merchant's PSAM and Merchant Acquirer (MA). The normal path directly between MA and M transfer of transaction batch (TTB) is shown in this Figure as D_TTBP. The alternate path is termed as A_TTBP. This path consists of two distinct protocols. A_TTBP1 is used between merchant's PSAM and MMA; A_TTBP2 is used between MMAPSAM and Merchant Acquirer. MMA with MMAPSAM becomes an additional entity in between. MMA collects transaction batches from Merchant through A_TTBP1 and MA collects transaction batches from MMA through path A_TTBP2.

The three protocols are discussed below:

D_TTBP

Protocol to transfer transactions is named here as "TTB" (Transferring Transaction Batch) shown in Figure. It is designed to transfer transactions in form of batch in secure manner from merchant's PSAM to MA, from any centralized location to MA or from an Internet terminal having smartcard reader attached to MA. This protocol is divided in three parts namely initialization, certificate verification and transaction transfer.

Initialization

Before transferring transactions from PSAM, it is necessary to initialize the card, where the data is stored. This initialization process will start by a terminal holding this card. Figure 3.2 shows the sequence of messages flowing between PSAM and terminal. After card initialization, terminal creates a secure channel or secure tunnel between card and communicating entity to transfer all subsequent messages flow between them.

After initialization of card by terminal, card is in state of accepting direct command coded for the application. Figure 3.3 shows the next subsequent messages. Next message to card coming from MA is 'Initialization of Transferring Transaction Batch' (Init_TTB). This message consists of initialization command for transferring transaction batch, which gives information to the PSAM about the transaction date, currency in use and location, country & domain of MA.

After receiving Init_TTB command from MA, PSAM does verification of the date and time of transaction with expiry date and returns information to MA for version verification. In response to Init_TTB, card returns Response to the Initialization Transferring Transaction Batch (R_Init_TTB) to MA.

Certificate Verification

Before starting the actual batch transfer from Merchant PSAM to Merchant Acquirer, both participating entities verify their certificates for authentication. Message flow for certificate verification is shown in the above diagram.

Transferring Transaction Batch

Actual transaction batch transfer begins after verification of certificate. Figure 3.4 shows the sequence of the message flow between PSAM and the Merchant Acquirer after verification of certificate for authentication. After verifying certificate by PSAM of MA, MA sends 'TTB'.

Message to PSAM consisting of:

TTB = (TTB Command, ID_{MA}, {{ID_{MA}, $SESSKey_{MA}$} SK_{MA}} PK_{PSAM})

After receiving TTB, PSAM decrypt the digital signature by his private key SK_{PSAM} to get the session key and sender's identity. Here, MA authenticate it self to the PSAM.

In response with this command TTB, PSAM prepares the transaction batch (TB) by appending each purchase transaction data (TD) with S5 MAC shown

in Figure 3.5. PSAM also creates Batch Summary (BS) record for total transactions in the batch and creates MAC S4 of the Batch summary record in a similar fashion. Final S7 is created for the encrypted batch.

Following gives details of R_TTB message:

$ID_{UB} = (ID_{PSAM} || ID_{BATCH})$

TB = (TD1, S51), (TD2, S52)… (BS, S4)

$ETB = (\{TB, ID_{UB}\}SESSKey_{MA})$

$R\_TTB = R\_TTB, (ETB, ID_{UB}), S7$

After verification of S4, S5, S7 MAC (Message Authentication Code), MA prepares the acknowledgment message (ACK_MSG) for the PSAM containing the acknowledgement of the current batch; Where ST is the status code.

$ACK\_MSG = ACK, \{ID_{UB}, ST\} SESSKey_{MA}, ID_{UB}$

A_TTBP1

A_TTBP1 is the protocol path between PSAM and MMA having MMAPSAM in one slot for authentication. The protocol description is same i.e. consisting of three steps, Initialization, certificate verification and transferring transaction batch. The first two steps remains the same pertaining to participating entities. The protocol differs only in the third step. MMA behaves as MA and initiates transaction. Message sequence is same as shown in Figure 3.4 and consists of;

$TTB = (TTB, ID_{MMA}, \{\{ID_{MMA}, \#RND\}SK_{MMAPSAM}\}PK_{PSAM})$

TB = (TD1, S51), (TD2, S52)… (BS, S4)

ETB1 = {TB, IDUB} SESSKeyPSAM

R_TTB = R_TTB, ETB1, {SESSKeyPSAM} SKPSAM} PKMA, IDUB, f (#RND), S8)

MMA sends back an acknowledgement message ACK_MSG containing status code as TEMP (to indicate Temporary Acknowledgement) along with the ID_{UB} to the PSAM's of PSAM and Batch.

$ACK\_MSG = ACK, ID_{UB}, ST$

A_TTBP2

MMA collects transaction batches from many merchants and stores these in his own data store (MMAPSAM) along with the ID_{UB}. It is noted here that each transaction batch is in encrypted form. This is to avoid any security issue related to MMA. Finally he transfers all batches to the MA. Protocol A_TTBP2 is used to transfer the transaction batches to MA.

MMAPSAM here behaves if it was a PSAM and initiates the transaction. All three steps of this TTB flow are repeated NNT times to transfer all transaction batches from MMA, where NNT is the number of transactions.

$TTBi = (TTB, ID_{MA}, \{\{ID_{MA}, \#RND\} SK_{MA}\} PK_{MMAPSAM})$

$R\_TTB1\text{-}NNT = R\_TTB, ETB1, \{SESSKey_{PSAM}\} SK_{PSAM}\} PK_{MA}, ID_{UB}, f (\#RND), S9$

TTBI is the ith batch TTB and S9 MAC is generated by the MMA for verification of data integrity by MA. After verification, MA prepares the acknowledgment for this batch and encrypts it with the session key which he has received from PSAM along with this batch, and sends ACK_MSG to MMA for each batch so received.

$ACK\_MSG1\text{-}NNT = ACK, ID_{UB}, \{ID_{UB}, ST\} SESSKey_{PSAM},$

MMA, after receiving this acknowledgement stores it in its data store (MMAPSAM) and deletes the corresponding transaction batch. This transaction batch transfer process continues till all transaction batches are transferred from MMA to MA, that is NNT batches.

The simulation and verification of protocol is given in [9] using SPIN. The security properties are not possible to verify using same tool. These can be done using the evolving tool AVISPA. The security properties verification using the mentioned tool is shown in the following section.

4. What is AVISPA

The AVISPA is an acronym for Automated Validation of Internet Security Protocol and Applications, is a Push-button security protocol analyzer [1], and supports the specification of security protocols and properties by means of a modular and expressive specification language. It integrates different back-ends implementing a variety of automatic analysis techniques for protocol falsification by finding an attack on the input protocol, and abstraction-based verification methods both for finite and infinite numbers of sessions. The user interaction is facilitated by a web interface that is an easy way to use AVISPA tool without installing any other software to support it. The next section describes the architecture of AVISPA tool.

4.1. AVISPA architecture

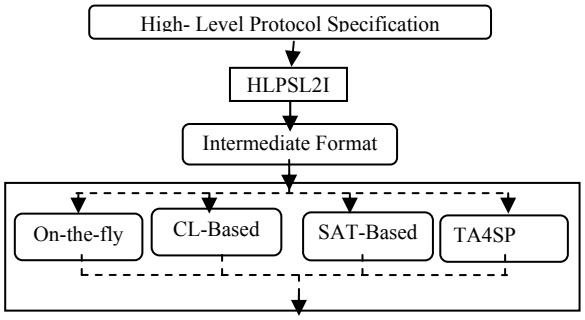

Figure 3 AVISPA Tool: Architecture

The architecture of the AVISPA tool is [19] shown in Figure 3. HLPSL-High Level Protocol Specification Language provides a high level of abstraction and has many features that are common to most protocol specifications such as intruder models and encryption primitives. The Intermediate Format (IF), the language into which HLPSL specifications are translated, is a lower level language at an accordingly lower abstraction level. HLPSL specifications are translated into the IF by the HLPSL2IF translator. These translations in turn, serve as input to the various back-ends. These are analysis tools of the AVISPA tool-set.

The High Level Protocol Specification Language (HLPSL): is an expressive language for modeling communication and security protocols. HLPSL draws its semantic roots from Lamport's Temporal Logic of Actions (TLA). TLA is an elegant and powerful language which lends itself well to specifying concurrent systems. The development of HLPSL was thus undertaken with the following design objectives:

- It must provide a convenient, human readable and easy to use language yet be powerful enough to support the specification of modern Internet security protocols. To achieve, HLPSL has been defined in such a way as to closely resemble a language for defining guarded transitions within a state-transition system and is equipped with constructs which allow the modular specification of protocols.
- It must be consists of a formal semantics. For this, HLPSL has been based on Lamport's TLA and its semantics is given by a translation to a subset of TLA
- It must amenable to automated formal analysis. This is achieved by a translation of HLPSL into the Intermediate Format.
- Supports symmetric and asymmetric keys, non-atomic keys, key-tables, Diffie-Hellman key-agreement, hash functions, algebraic functions, typed and untyped data, etc.
- Supports security properties, different forms of authentication and secrecy.
- The intruder model is made by the channel(s) over which the communication takes places.
- Role based language, a role for each (honest) agent and Parallel and sequential composition glue roles together.

The HLPSL2IF translator automatically translates a HLPSL protocol specification provided by the user into an IF specification, which is then given as input to the different back-ends of the AVISPA tool. Hence, the main goal in the design of the IF was to provide a low-level description of the protocol that is suitable for automatic analysis and yet this format should be independent from the analysis methods employed by the various back-ends.

The Back-Ends are used to provide protocol falsification, bounded and unbounded verification.

The On-the-fly Model-Checker (OFMC) employs several symbolic techniques to explore the state space in a demand-driven way. It builds the infinite tree defined by the protocol analysis problem in a demand-driven way i.e. on-the-fly, hence the name of the back end. It uses a number of symbolic techniques to represent the state-space. OFMC can be employed not only for efficient falsification of protocols i.e. fast detection of attacks, but also for verification i.e. proving the protocol correct for a bounded number of sessions, without bounding the messages an intruder can generate.

CL-AtSe (Constraint-Logic-based Attack Searcher) applies constraint solving with

simplification heuristics and redundancy elimination techniques. It provides a translation from any security protocol specification written as transition relation in the IF, into a set of constraints which can be effectively used to find attacks on protocols. Both translation and checking are fully automatic and internally performed by CL-AtSe, i.e. no external tool is used.

The SAT-based Model-Checker (SATMC) builds a propositional formula, encoding a bounded unrolling of the transition relation specified by the IF, initial state and set of states representing a violation of the security properties. The propositional formula is then fed to a state-of-the-art SAT solver and any model found is translated back into an attack.

TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols) approximates the intruder knowledge by using regular tree languages and rewriting to produce under and over approximations. The TA4SP tool, computes either an over-approximation or an under approximation of the intruder knowledge by means of rewriting on tree languages in a context of unbounded number of sessions. An over-approximation may lead to positive proofs of secrecy properties on the studied protocol for an unbounded number of sessions. In an under-approximation context, without any optional abstractions, the tool may show that the protocol is flawed for a given secrecy property.

4.2. Protocols Verified by Using AVISPA

AVISPA is the acronym for Automated Validation of Internet Security Protocol and Applications. The design purpose of the tool itself indicates its importance for verification of security properties. Many security protocols and systems are verified using AVISPA as given in [19]. Using the web interface any specification of the protocols are validated. Various security protocols along with their specification and flaws are discussed. Key Management Architecture for Hierarchical Group Protocols has been verified using AVISPA tool in [20]. Also the Cross-Layer Verification of Type Flaw Attacks on Security Protocols was detected as in [21].

4.3. Modeling of the protocol in AVISPA

The payment protocol is the outcome of the thesis [9], modeled using AVISPA tool for verifying its security properties. The messages involved in security are only taken into consideration. The protocol is modeled using HLPSL specification. Parties involved in the communication are modeled as the roles with their message exchanges. A session is created between the roles to achieve the required message exchanges. The session also considers an intruder in between. This is modeled using one session which includes the intruder, having the knowledge of keys of sender and receiver. Finally the environment is created for all the sessions simultaneously.

5. Security properties verification

AVISPA tool is mainly designed to prove the security properties. Some of the properties are verified as follows.

5.1. Analyzing Security Properties

The protocol satisfies the security properties like authentication, integrity and secrecy which are analyzed below:

Authentication

Authentication is the process by which both the participants know each other and guarantee that they are communicating with the desirable party. Three parties are involved in this protocol, Merchant-M, Merchant Acquirer-MA and Mobile Merchant Acquirer-MMA. Mutual authentication is established in the protocol TTB or D_TTBP in the following way. After exchanging and verifying certificates, MA sends encrypted digitally signed $SESSKey_{MA}$ to PSAM (Personal Secure Authentication Module) of M to authenticate itself to PSAM. PSAM then sends transactions encrypted by received session key and authenticates itself to MA. This can be modeled in AVISPA considering Ks as the session key;

MA → M; `SND(MA.Kma) /\ RCV(M.Km')`
MA → M; SND ({{MA.Ks} _ (inv (Kma))} _Km)

In A_TTBP also, mutual authentication needs to be done between PSAM and MA. PSAM creates a the session key $SESSKey_{PSAM}$, M signs this session key along with ID_{PSAM} with his secret key and then encrypts it with MA's public key. This encrypted digitally signed session key Ks and ID are sent to MA through MMA.

M → MMA ;
`MSG':= {{Ks'}_(inv (Km))}_Kma`

```
 /\SND(ETB1'.MSG'.IDUB'.Nm'.S8')
```
MMA → MA;
```
 SND(ETB1'.MSG'.IDUB'.Nmma'.S9')
```

As this is signed by PSAM, PSAM authenticate itself to MA. MA in return, encrypts the acknowledgement with same session key, thus MA authenticate itself to PSAM.

MA→MMA; `SND (Ack_F'.IDUB'.A1')`
MMA→M `SND (Ack_F'.IDUB'.A1')`

`A1'` is the encrypted message by the session key. In D-TTB it's a direct acknowledgement from MA to M using Session key.

MA→M;
`SND(ACK'.IDUB'.{IDUB'.ST'}_(inv(Ks))`
`)`

We can easily show that this indirect authentication cannot be tampered by the intermediary MMA. Firstly, the session key sent by PSAM cannot be deciphered by MMA since this action needs the secret key of MA. Similarly, the ACK_MSG sent my MA cannot be deciphered by MMA since this action will again need the session key. Since we need to combine two protocols A_TTBP1 and A_TTBP2, the resultant one is A_TTBP in which all three parties are communicating with each other but with different sessions. Also the individual protocols A_TTBP1 and A_TTBP2 protocols are modeled and tested differently.

Mutual authentication is done in protocol A_TTBP1 and A_TTBP2 with the exchange of nonce. In A_TTBP1, MMA sends nonce digitally signed by secret key of MA and encrypted with PSAM's public key, which authenticates MMA itself to PSAM, PSAM in return sends a function of nonce back to MMA to complete mutual authentication.

MMA→M;
SND({{MMA.Nmma'}_(inv(Kmma))}_Km')

Where Nmma is the nonce generated, Kmma, Kma and km are the public keys of MMA, MA and M respectively. Same thing is done in A_TTBP2, where MA sends encrypted digitally signed nonce to MMAPSAM and in return MMAPSAM returns function of nonce back to MA for mutual authentication.

MA→MMA;
SND({{MA.Nma'}_(inv(Kma))}_Kmma')

In all the above messages nonce are used to indicate freshness of the message.

Secrecy

This property implies that the information is not made available or disclosed to unauthorized users, entities or processes. This protocol meets the requirement by sending transaction batches to MA from MMA encrypted with session key. In direct path TTB or D_TTBP, MA generate the session key $\mathbf{SESSKey_{MA}}$ and shares it with PSAM after digitally signing and encrypting with PSAM's public key. This session key can only be decrypted by PSAM and is known only to PSAM and MA. Hence secrecy of transaction batch is maintained in the protocol path TTB or D_TTBP.

Similarly, in path A_TTBP PSAM generate the session key $\mathbf{SESSKey_{PSAM}}$, shares it with MA after digitally signing it with his secret key and then encrypting it with MA's public key. Transaction batch is encrypted by this session key and sent to MA through MMA along with encrypted digitally signed session key. Only MA can decrypt this session key as it is encrypted by his public key. Hence the secrecy is maintained in this path also.

Protocol maintains the secrecy of acknowledgement also. In protocol path TTB, ACK_MSG returns to PSAM encrypted by session key used in that session to transfer the transaction batch. i.e. encrypting ASK_MSG using $\mathbf{SESSKey_{MA}}$. Similarly in alternate protocol path A_TTBP ACK_MSG is encrypted using $\mathbf{SESSKey_{PSAM}}$ by MA, which is not known to MMA.

Integrity

This is the property wherein information has not been altered or destroyed in an unauthorized manner. The protocol proposed here also maintains this property. In D_TTBP, when merchant's PSAM need to send the transaction batch M prepares the message R_TTB consisting of encrypted transaction batch, identity of merchant and identity of batch, then creates hash of the above data called as message authentication code (MAC) S7. This S7 MAC is also sent with above data to MA. MA receives all above data and creates hash with the same algorithm and verifies the received S7 MAC. When MA finds these two MACs are same, it confirms that data integrity is not violated. Similarly, MAC S8 permits MMA to ensure data integrity of the message received from PSAM. S8 is computed over the entire message containing the ETB. Thus, while MMA ensures integrity of the received message, it cannot still

decrypt the individual components of the message. Similarly, MAC S9 is used for ensuring integrity by MA of the message sent by MMA.

The MAC S5 and S4, generated by PSAM at the time of purchase transaction and at the time of batch formation, are securely transferred to MA as these form part of the ETB which cannot be deciphered by MMA. These MACs are used by MA to verify the integrity of the individual transactions and transaction batches. AVISPA code to provide integrity for one of the MAC S7 is given as:

M → MA: S7':= H (IDUB'.ETB')

5.2. Analyzing Attacks on Protocol:

We have considered the common type of attack on the protocol

Reuse of Confidential Data Attack

In this protocol confidential data is digitally signed by the secret key of the sender and then encrypted it by the receiver's public key. For example, in protocol path D_TTBP message 3.

MA → PSAM: TTB, $\mathbf{ID_{MA}}$, $\{\{\mathbf{ID_{MA}}, \mathbf{SESSKey_{MA}}\}_{SKMA}\}_{PKPSAM}$

Here, reuse of confidential data attack cannot occur, as $\mathbf{SESSKey_{MA}}$ is first digitally signed by the secret key of MA ($\mathbf{SK_{MA}}$) and then encrypted with the public key of PSAM ($\mathbf{PK_{PSAM}}$). In AVISPA it is modeled as;

MA → M; SND ({{MA.Ks} _ (inv (Kma))} _Km)

Similarly in other protocols A_TTBP1, and A_TTBP2, whenever there is need to encrypt for authentication we are signing it first and then encrypting the data with receiver's public key. This ensures that the protocols are flawless on this count.

Replay Attack

In case of TTB or D_TTBP intruder captures all the messages from MA to PSAM and afterwards replays it posing himself as a PSAM. This may transfer the same batch to MA, which is detected at MA because MA keeps log of batch summary record of every received batch.

Similarly, intruder may captures all message from MA to PSAM and replay it to PSAM posing himself as a MA. This may transfer the batch to intruder but intruder cannot replay the ACK_MSG as this contains the identity of batch (ID_{BATCH}), that is encrypted by session key which intruder cannot get.

In alternate path protocol A_TTBP1 and A_TTBP2 the nonce for authentication is used, which is a random number generated only once. Here also replay is not possible. Hence this protocol is foolproof against the replay attack.

Life of Session key

Session key cannot be compromised, because this protocol makes use of online communication and thus the life of key is very short. But in case of alternate protocol path, life of session key that is used to encrypt the transaction batch, is more. There are more possibilities of compromising session key in this path. This is the only weakness in alternate path. However considering the value of the transaction batch which is likely to be of the order of Rs 300 to Rs 1000 only, it is not likely that serious attack will ever be mounted to disrupt payment. No financial gain can remit for the attacker since the money is paid only to know merchants by the acquiring bank.

6. Implementation and Verification Results

The implementation consists of modeling of the protocol using HLPSL of AVISPA. The above attacks and properties are modeled. The verification results are given below.

D-TTBP

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 /root/avispa-
1.1/testsuite/results/pr_5-1.if

GOAL
 As Specified

BACKEND
 CL-AtSe
```

```
STATISTICS
 Analyzed : 3843 states
 Reachable: 2724 states
 Translation: 0.01 seconds
 Computation: 1.33 seconds
```

A-TTBP

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 /root/avispa-
1.1/testsuite/results/pr_5-6.if

GOAL
 As Specified

BACKEND
 CL-AtSe

STATISTICS

 Analyzed: 325 states
 Reachable: 325 states
 Translation: 0.17 seconds
 Computation: 0.07 seconds
```

By referring the other security protocols that are already verified using AVISPA, a conclusion is drawn that the protocol is safe. It indicates there is no authentication attack and secrecy attack on the protocol. The authentication of the participating parties is done by exchanging their certificates. There is no attack found on session key by the intruder. Also secrecy of session key and transferred messages between parties are maintained. By including hash function in modeling, the integrity is also maintained. The backend itself is the attack searcher that gives no information about the attack, indicating protocol is safe. By referring TLS (Transport Layer Security) protocol, as more number of state are analyzed and reachable, the protocol is safer. Thus, three security properties are modeled and verified using AVISPA tool.

7. Analysis of results

A new tool AVISPA is used to verify the security properties like secrecy, integrity and authentication. It gives details about whether protocol is safe or not. If not then it also gives the trace of the attack found, to indicate secrecy attack or authentication attack. So even though many properties of the protocol are to be checked, but only few can be verified using AVISPA. For this the modeling is done in HLPSL, language used by AVISPA tool. This modeling can be verified with any of the four back-ends of the AVISPA. Secrecy, integrity and authentication are verified using this tool. For our verification, we have used cl-Atse to search for the attacks on the protocol.

The feature of finding and tracing the attack makes AVISPA different than the other tools. Hence the tool is tested and the protocol verification results are analyzed**.**

8. Conclusion

The protocol used for transferring transaction batches from Merchant to Merchant Acquirer is proposed in [9] by one of the author, is verified for its correctness using SPIN. And same protocol is verified for its security properties using AVISPA.

Time consuming and costly testing can be reduced using formal verification, and any such system which is being accessed either by one or more parties, can be fully checked against their specification. A flaw or error can also be traced and verified accordingly. A detailed protocol model has been worked out for implementing the system, wherein a batch of payment transactions is securely transferred at the end of day to the back-end settlement system used by corresponding Merchant Acquirer. The security properties are analyzed using the tool AVISPA. It is shown that the protocol security holds within the established security of PKI. As more and more such protocols are evolving, validation and verification of such protocols become easier with the help of AVISPA tool.

9. References

[1] Armando1 and others, The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications, CAV 2005, LNCS 3576, pages. 281–285, 2005

[2] Schneider S. and others, verifying Authentication Protocols with CSP, Proceedings of the IEEE Computer Security Foundations Workshop X, pages 3-17, IEEE Computer Society Press, 1997.

[3] Meadows C., The NRL Protocol Analyzer: an overview Journal of Logic Programming, February 1996.

[4] Mitchell J.C. and others, Automated Analysis of Cryptographic Protocols Using Murphi, Proceedings of the 1997 IEEE Symposium on Security and Privacy (1997) pages 141-151, IEEE Computer Society Press.

[5] Roscoe A. W., Modeling and verifying key exchange, protocols using CSP and FDR, In Proceedings of 8th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, 1995.

[6] University Of Cambridge, http://www.cl.cam.ac.uk/research/hvg/Isabelle/overview.html, updated 12-07-2006.

[7] Giampaolo Bella and others, Verifying the SET Registration Protocols, IEEE Journal on Selected Areas in Communications, vol. 21, no. 1, pages 1-14, January 2003.

[8] AVISPA Tool Documentation, IST-2001-39252, downloaded from http://www.avispa-project.org/publication.html/, 2002. Accessed march 2009.

[9] S. R. Devane, Technical Challenges in Smart Card based Indian Payment System for Limited-Connectivity Environments, Thesis report, IIT Bombay, pages 1-149, January 2006.

[10] Madanlal Musuvathi, CMC- A model checker for network protocol implementation, Thesis report of Stanford University, Dept of Computer Science, February 2004.

[11] Yuji kukimoto, VIS Technology Transfer course, Session 3, updated on 09-26-2006.

[12] Prof. Daniel Kroning, Work-Shop 2006, Dept Computer Science, ETH Zurich, http://www.inf.ethz.ch/~daniekro/, 2006.

[13] Jim Huggins, Abstract State Machines, http://www.eecs.umich.edu/gasm, downloaded on Jan 2007, updated 29 September 2006.

[14] B-Core (UK) Limited, Kings Piece Harwell, Oxon Ox11 OPA, UK, downloaded from http://www.b-core.com, updated 22 February 2002.

[15] Clark and Wing, Formal methods: State of the art and future directions, ACM Computing Surveys, Vol. 28, No. 4, pages 1-18, December 1996.

[16] Roscoe A. W., Modeling and verifying key exchange, protocols using CSP and FDR, In Proceedings of 8th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, 1995.

[17] Xiang Fu and others, WSAT: A Tool for Formal Analysis of Web Services, downloaded from www.cs.ucsb.edu/~su/papers/2004/CAV2004.pdf, pages 1-4, published 2004.

[18] On The Fly, LTL model checking with SPIN, downloaded from http://SPINroot.com/SPIN/, updated 23rd June 2006.

[19] AVISPA Tool Documentation, IST-2001-39252, downloaded from http://www.avispa-project.org/publication.html/, 2002.

[20] Mohamed Salah Bouassida and others, Automatic Verification of a Key Management Architecture for Hierarchical Group Protocols, LORIA, Campus scientifique, B.P. 239, 54506 Vandoeuvre-l`es-Nancy Cedex – France, pages 1-15, 2006.

[21]Benjamin W. Long, Cross-Layer Verification of Type Flaw Attacks on Security Protocols, 30th ACSC-2007, Vol-62, Australia, pages 1-10, 2006.

