Automated verification tools for cryptographic protocols

1st Adel Hassan
Information Technology Engineering
Al Quds University
Jerusalem, Palestine
adel7377@gmail.com

2nd Isam Ishaq
Information Technology Engineering
Al Quds Universityna
Jerusalem, Palestine
isam@alquds.edu

3nd Jorge Minilla
Telecommunication Engineering
University Of Malaga
Malaga, Spain
munilla@ic.uma.es

***Abstract*—Security protocols are used to secure communications using cryptographic tools. Different communication and network security protocols, such as HTTPS and TLS, have become widely used in current applications and systems that are deployed in different industries such as banking, insurance, mobile operator, etc. Verifying the security of these security methods is becoming increasingly important to guarantee that they fulfill their goals, and automated verification tools are profiled as the most promising instrument for this task. Automated verification tools can be categorized according to the approach that they use, symbolic or computational, and their prove method: model checking and theorem prove. This article aims to review the methods employed by the different types of verification tools and provide a comparative of the tools more widely used nowadays.**

***Index Terms*—Verification tools; security protocols; cryptography; formal analysis**

I. Introduction

Security protocols and cryptographic primitives aim to protect systems by securing the communications and networks. The pyramid of security architecture depends on three components: Confidentiality, Integrity, and Authentication (CIA). Attackers or adversaries could try to exploit the weakness of these protocols' design using different methods of attacks. These attacks have different types and forms aiming to break any or several of the CIA goals. A few examples of such attacks are:

- Man-In-The-Middle (MITM) attacks: attackers stands between the two legitimate parties and relays and manipulate the messages exchanged between them.
- Replay attacks: Attackers capture messages and reply on behalf of the correspondent parties (sender and receiver).
- Confusion attacks: Attackers can modify messages with the specific key in the protocol's body.

An exhaustive analysis is needed to deal with such manipulations and improve cryptographic designs to harden the architecture. Thus, the use of automated tools to verify formally the security of the protocols is becoming crucial. The formal verification implies the application of mathematical notation and proofs for rigorous specification, design and verification of the systems. Automated verification tools, as shown in Fig. 1 [1], use two major formal approaches, symbolic and computational, and two methods, model checking and theorem proves.

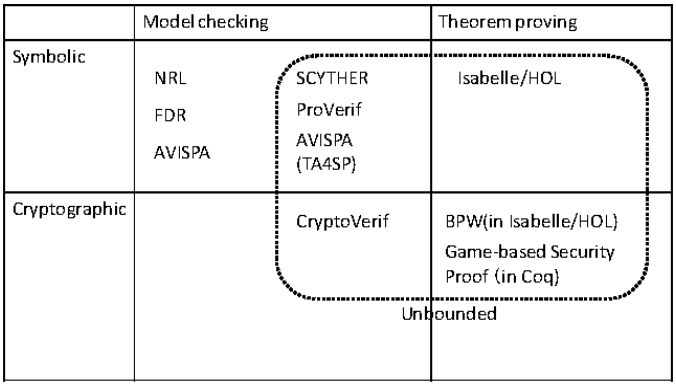

Fig. 1. Formal Methods Categorization for cryptographic protocol analysis[1]

These approaches identify challenges coming from several applications, and try to develop new models and tools such as the following [2]:

1) Develop models and tools that can be used in all applications achieving most security and privacy of using technologies from hardware perspectives.
2) Develop models to deal with all types of attacker's (human or machine).
3) Develop technologies that permit mathematical proofs to be implemented. (computational soundness and full abstraction).
4) Develop tools to ensure information and infrastructure security and prevent different attacks at infrastructure system levels.
5) To ensure and enhance the usability of tools used in such types of analysis.

The rest of this paper is organized as follows; Section II describes the history of formal methods. Section III clarifies the review of existing tools for formal verification of cryptographic protocols. Section IV is focused on the application of the protocol specification and verification. Section 5 V compares the different Automated Verification Tools. Finally, Section VI draws the main conclusions and foresees future works.

II. Related Works

Dolev and Yao [3] developed the first formal model (DY) using different cryptographic algorithms, encryption and decryption operations. They used state-space exploration techniques to find out if a final vulnerable state was reachable or not. If it was, then the attack could reach its goal and it was considered successful [4]. [5] introduced new logic called BAN logic, which is different from the DY approach, based on the logic of belief, which describes the relationship between principles and data. [6] used failure divergences refinement as a general-purpose checking model. [7] added another mechanism by using communications sequential processes language to analyze security protocols of a formal model. After these, a myriad of verification tools have been proposed, which can be divided depending on their verification approach and method used.

According to the verification approach, we have:

- Computational Approach (CA): a protocol is considered secure when all of its components are secure. The attacker tries to break primitives of the encrypted protocols with polynomial-time boundaries using Turing machines, so further reduction is needed and cannot be done without proofing and verification by hand. This model can cover only logical attacks but not hardware attacks. However, the computational model is considered a compelling model that guarantees security aspects of security protocol, but it is still very complex when trying to automate the model's proof. Examples of tools that follow this approach are EasyCrypt [8], Cryptoverif [9], and F* [10].
- Symbolic Approach (SA): the assumption here is that cryptography is ideal and complete and meets with threat model [3]. The attacker goals are represented as mathematical equations and then automated checking is applied to ensure that the goal of the attacker is not achievable. SA works at the abstraction level, and it has been used, for example, to describe the flaws in Gmail (sign-in browser ID) . On the other hand, the symbolic approach deals with an individual protocol and does not deal with how cryptographic protocols result when working simultaneously with others (communication infrastructure). Multiple fully automated verification tools use the symbolic approach, such as Proverif [11], Scyther [12], Maude-NPA [13], and AVISPA [14]. There is also other tools used as information control and testing tools that use this approach, such as Cassandra [15], Dafny [16], SAGE [17] and Tamarin [25].

The main difference between these approaches is that SA describes what an attacker can do, while CA describes what an attacker cannot. Both approaches have advantages and disadvantages, but basically we could say that CA is more realistic but also more complex, while SA can be more easily simulated but assume ideal cryptography, which is less realistic.

According to the proving method, most of verification tools can be divided into:

- Model Checking: this model is inspired by the Kripke structure encoded by Boolean formulas as a form of algorithmic verification. Both secrecy and authentication in the security domain can be reduced to reachability problems [1] when state space is finite. There are many tools that employ model checking (CA or SA), such as NRL, AVISPA, Proverif, Scyther, and Cryproverif.
- Theorem Proving: this model uses reduced or inductive verification, instead of algorithmic verification, to prove the theorem in both first and higher-order logic by looking at the events of communications between protocol principles and intruder. This model needs arduous work and experts to continue analyzing and verifying primarily when the protocol classifies as an unbounded model.

Moreover, different improvements and modifications to the above described approaches and methods have been described in the bibliography.

- Computational soundness and full abstraction: these techniques simplify reasoning systems and solve weaknesses in different approaches and models. Complete abstraction [5] mostly uses programming language methodologies to achieve security of implementation of programming languages (language safety). Computational soundness try to guarantee that there are no existing bugs in the used tools that lead to non-expected results. It measures the maturity of the used model.
- Model Extraction and Code Generation [18]: Fig. 2 sketches these approaches that deal with application code instead of libraries of protocol logic. Model extraction is based on classical theory of abstraction to implement and prove security protocol properties in a simple manner. Code generation is implemented using model-driven software. Despite that, the model needs to take the computational model into account, and still there is no easy way to do it.
- JavaSPI: also introduced by [18] as an implementation language model using Java language (see Fig. 3). However, it still suffers from slowness in code generation.

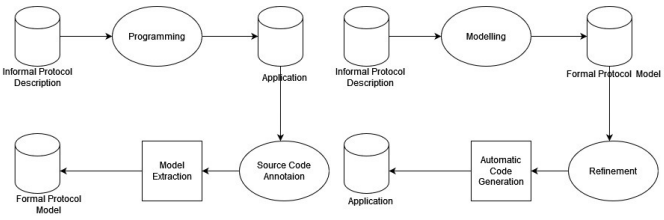

Fig. 2. Typical workflows used by a) model extraction; b) code generation [18]

III. Review of Tools for Formal Verification:

This section describes some tools that are used for both symbolic and computational approaches to automate analysis and verification of cryptographic and security protocols by generating executable code from specifications to verify if it is secure or not (security guarantee).

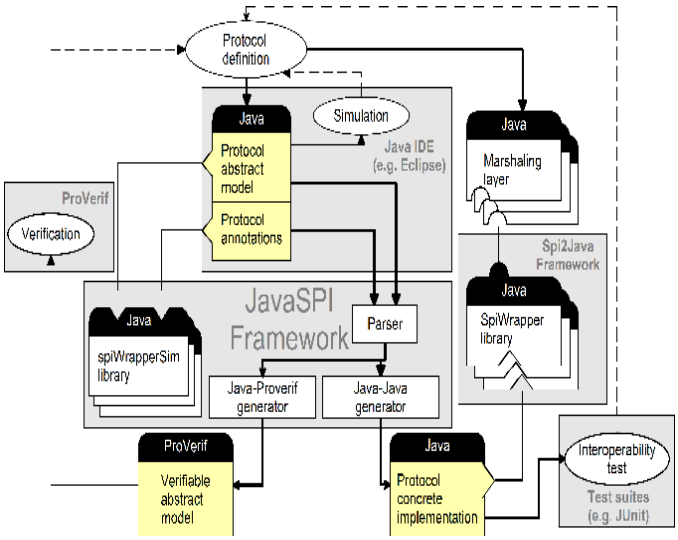

Fig. 3. JavaSPI Architecture [18]

1) **Interrogator:** it has a GUI interface and is implemented into the LISP machine described by [19]. It is used to determine the vulnerabilities and flow of key distribution in security protocols that are defined as text format and controlled by Lisp machines. The attacker can be shown on the display of the tool to simulate attack scenarios. If the attacker manages to exploit any weakness of the protocol, the system will popup messages that contain the changes that the attacker manages to do.
2) **Inatest:** it considers a type of symbolic method using formal verification approaches to generate attacks to cryptographic protocols. These attacks target networks and rules of security protocols described by Ina [20], using assertion language and specific algebraic properties. The analysis of the security protocol is still made in a manual way without the need for automated algebraic hop (Reduction).
3) **NRL Protocol Analyzer:** it is built and written in Prolog to analyze and specify Navel's laboratory [21]. The tool depends on a structure based on the DY threat model as shown in Fig. 4 and uses a symbolic approach for representing the state machine.

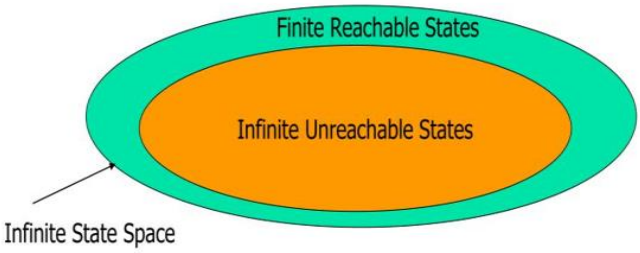

Fig. 4. NRL Analyzer

4) **Proverif:** this tool [22] is used to analyze and verify correspondence in the cryptographic protocols. It uses the mathematical model Pi calculus (language for modelling security protocols for studying concurrency and process interaction) and logic programming for cryptographic primitives and rule-based solver for the calculus matter. This tool uses the theorem prove model and has no bounds. It uses a non-exponential formula to analyze the adversary behavior at the network. Sometimes the algorithm enters an infinite loop and cannot be terminated when using the exponential equation. Figure 5 shows the verification methods in Proverif tool.

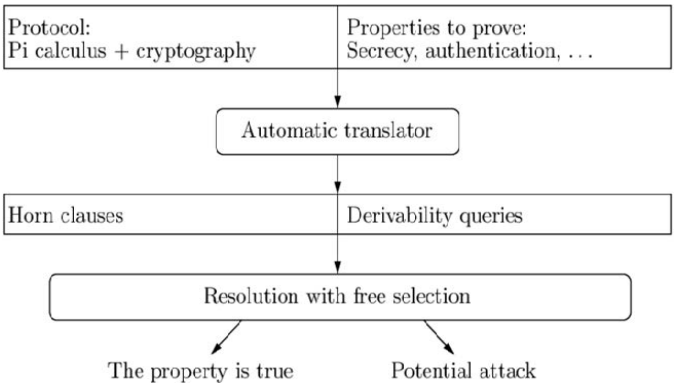

Fig. 5. Verification Method in Proverif

5) **Scyther:** Scyther [23] analyzes the security protocol by verifying the security claims. It uses a GUI for both analyzing and verification of security protocols, which uses a refinement algorithm and Security Policy Definition Language (SPDL) to describe the protocols. Figure 6 shows Scyther abstract structure and Fig. 7 displays Scyther editing tool.

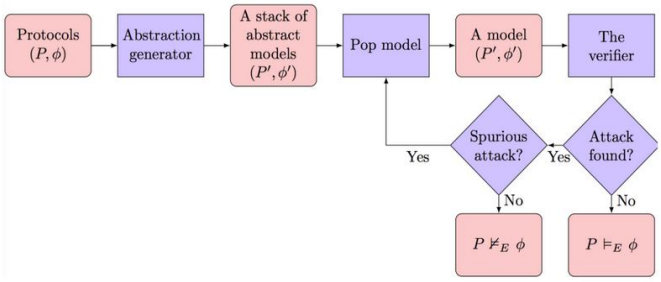

Fig. 6. Scyther Abstraction Structure

6) **AVISPA:** It is considered an automated validation tool for security protocol using a GUI and uses formal high-level protocol specification language (HLPSL) to define protocol specifications and properties. AVISPA consists of four back-ends, which are [24]: on the fly model (OFMC) ; SAT based Model checker ; Constraint logic-based model checker and Tree automate-based automatic approximation for analysis of the protocol. These back-ends assume that the cryptography is at a perfect state and that the adversary controls the network based on the DY model. When the protocol terminates, each back-end

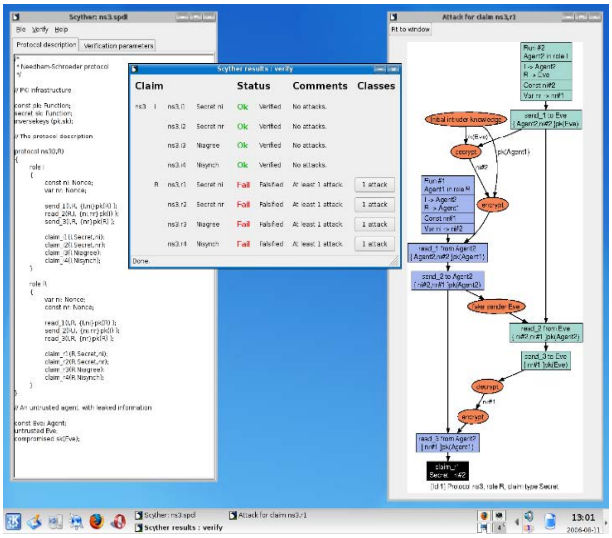

Fig. 7. Scyther Editor Tool

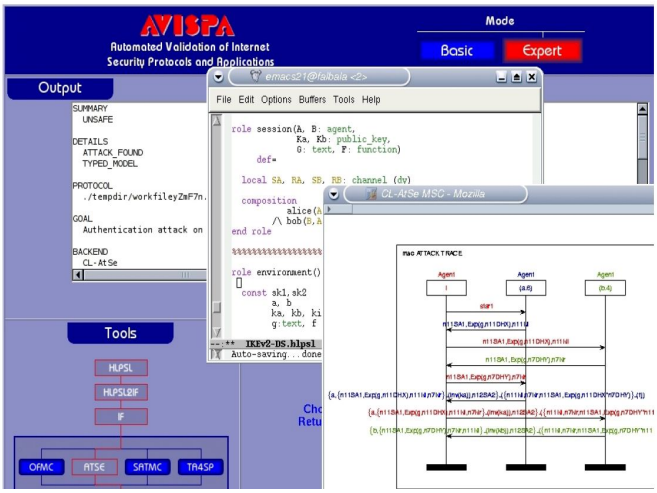

Fig. 9. AVISPA Editor Tool

outputs the result of analysis using an output format, which states whether the input format is solved, system resources are exhausted, or the problem is not tackled. Automated Validation of Trust and Security of Service-oriented (AVANTSSAR) is a further development of the AVISPA. Figure 8 shows AVISPA abstract structure and Fig. 9 displays AVISPA editing tool.

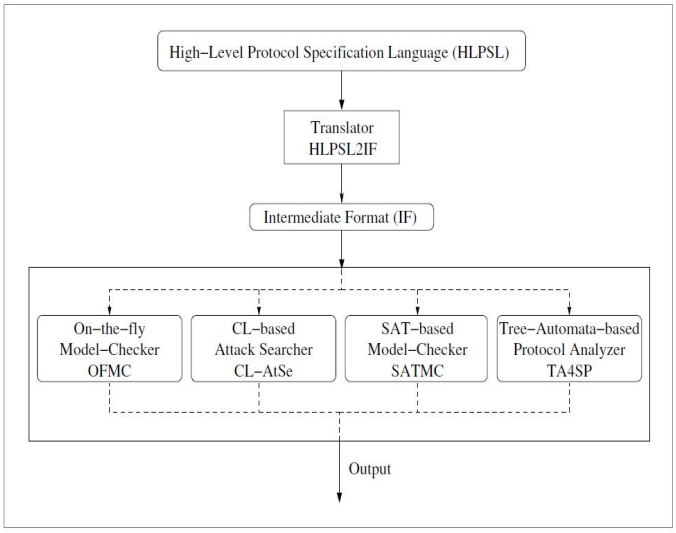

Fig. 8. AVISPA Abstraction Structure

7) **Tamarin Prover:** it is considered a symbolic type tool, that automates cryptographic protocols with an unbounded number of sessions. Tamarin [25] uses backward reasoning methodology and consists of two modes. The first mode is an interactive mode, and the second mode is a fully automated mode. It is written in the functional Haskel language, and considered the most used tool in analyzing and proofing security protocols. Moreover, it gives the user graphical attackers behavior and manual proof guidance in the whole process of attacks. Fig. 10 shows the editor of the TAMARIN tool with proof scripts and visualize processes of attacker.

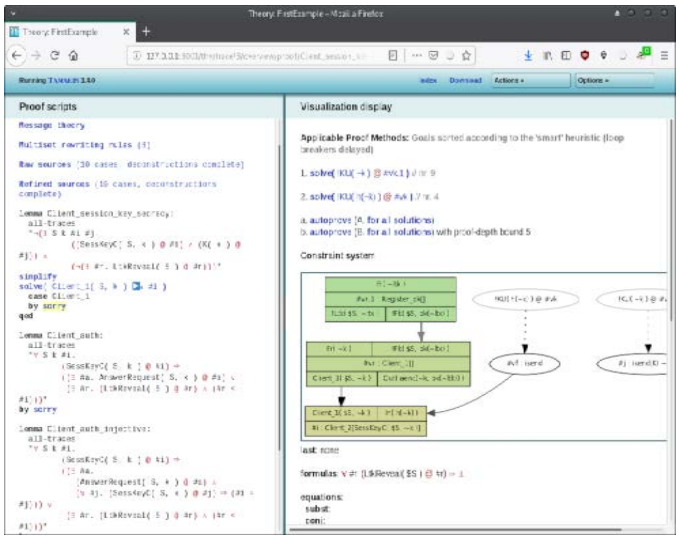

Fig. 10. TAMARIN Editor Tool

8) **COQ Theorem Prover:** [26] describes the tool as a calculus of inductive construction logical framework that enables a formal proof with functional programs. It is designed based on a mathematical model and writes a formal specification program to check and validate protocol format specifications correctness. The language used in this tool is called GALLINA and used to provide specification by using a type-checking algorithm to be sure that proof is correct. Fig. 11 shows the editor of the COQ tool with proof scripts and visualize processes of the attacker.
9) **Isabelle:** As described by [27], Isabelle uses formal interactive resonating with high order logic HOL. It is a

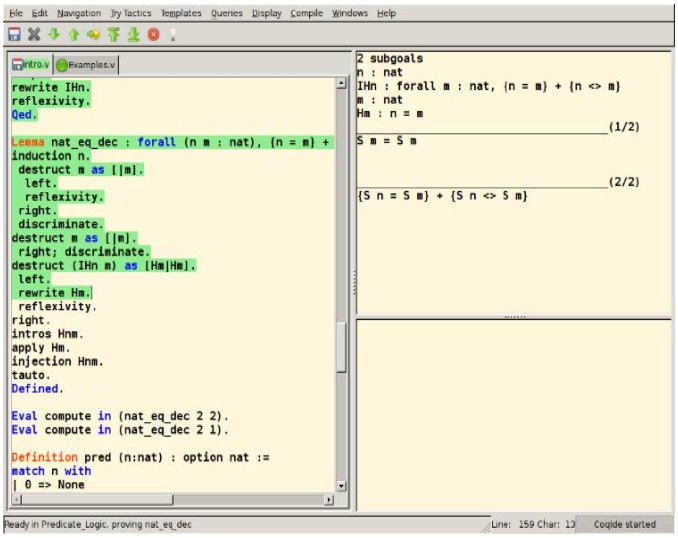

Fig. 11. COQ Theorem Prover Editor Tooll

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. It is considered complex to use if the user does not have enough knowledge of how to deal with the tool. Figure 12 shows the editor of Isabelle tool with proof scripts and visualize processes of the attacker.

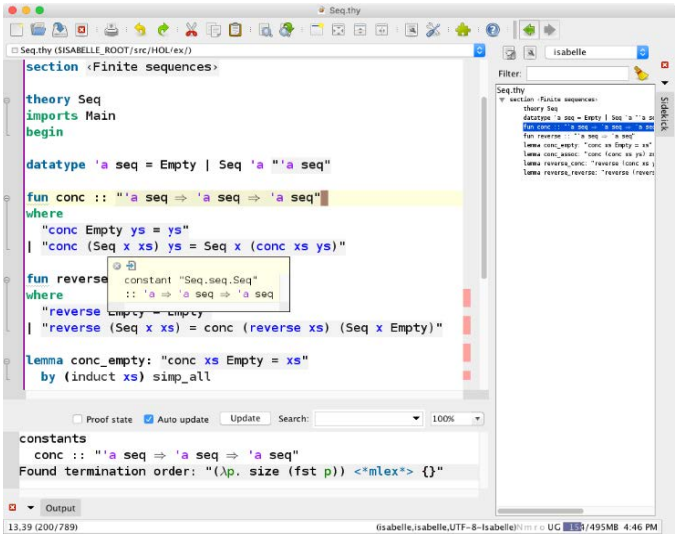

Fig. 12. Isabelle Editor Tool

10) **Murphy:** [28] introduced Murphy as a state enumeration tool to analyze security protocols. The Murphy language is used to check the model's reachability, comparing protocol specifications in the enumeration state. Murphy believes it is non-deterministic since it allows execution multiple times in the same process and supports the automatic symmetry hop (reduction)model. Figure 13 shows a sample of error log using the Murphy tool.
11) **Athena:** It is considered a fully automated tool to

```
CONST -- constant declarations
MAX_STATE_VALUE : 5;
TYPE    -- type declarations
state_type : 0 .. 10; -- integers from 0 to 10
disturbance_type : 0 .. 2;  -- integers from 0 to 2
VAR     -- (global) variable declarations
x : state_type;  -- x is a variable of type state_type

-- define next state function
function next(x: state_type; d : disturbance_type): state_type;
begin if (x <= 3) then return (x + d); else return (x - d); endif end;

startstate "init" x := 0; end;  -- define initial state

ruleset d : disturbance_type  do  -- define transition rule
   rule "time step" true ==> begin x := next(x, d); end;
end; -- ruleset d

invariant "x is not too big" -- define property to be verified
(x < MAX_STATE_VALUE);
```

Fig. 13. Murphy error log trace

analyze and verify security protocols [29]. Athena falls into both models, checking and theorem proving areas as described above. Herzog and Guttman proposed the standard space model SSM [30], which was used for protocol representation and demonstrated use for manually proving specific security properties. Figure 14 shows the editor of Athena tool with proof scripts and visualize processes of the attacker.

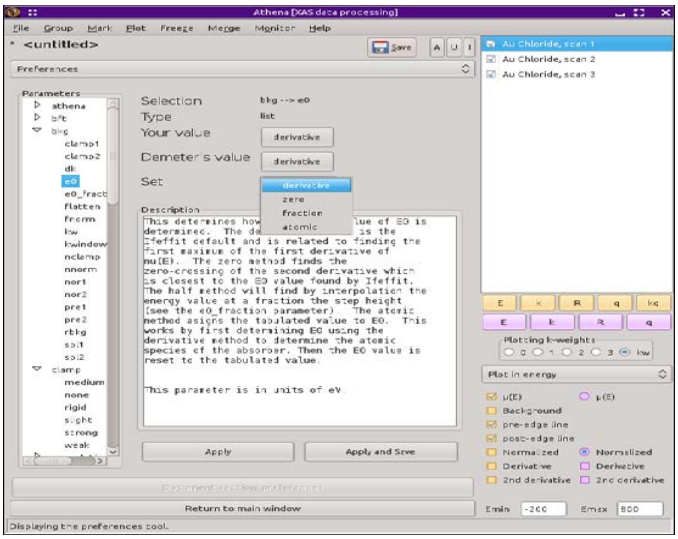

Fig. 14. Athena Editor Tool

12) **Brutus:** Brutus considers a special edition of the model checker to verify the properties of security protocols. Moreover, the algorithm used in this tool combines natural deduction and state-space exploration techniques. When the flow is discovered, the tool creates a counterexample using reduction techniques [31]. Figure 15 displays Brutus security protocol verification process.
13) **Casper:** it uses model checker as a compiler to analyze security protocols. [32] describes how the compiler uses a sequential algebra process. The user specifies the protocols using abstract notation, and Casper converts this into communicating sequential processes CSP code that is suitable for checking using the failures-divergence

Fig. 15. Brutus Verification Process

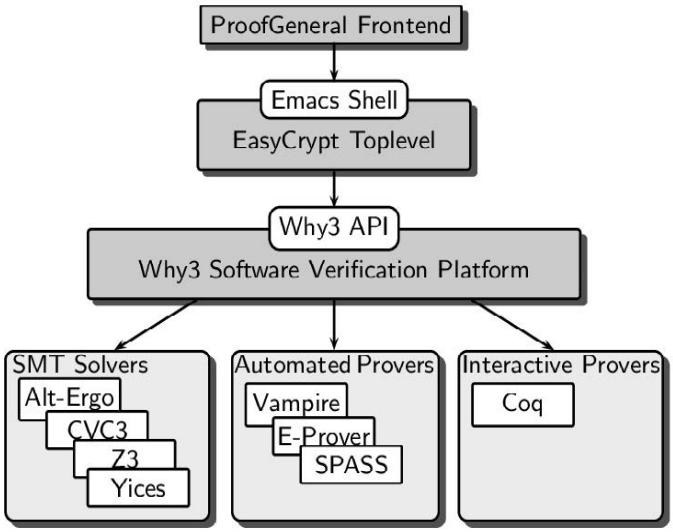

Fig. 17. EasyCrypt workflow tool

refinement FDR [33] that is used as a model checker. Casper does not cover all the features of security protocols, but it does provide a useful framework for investigating new ideas and considering how to model new features within CSP and can deal with a vast class of protocols. Figure 16 shows Casper verification results with FDR.

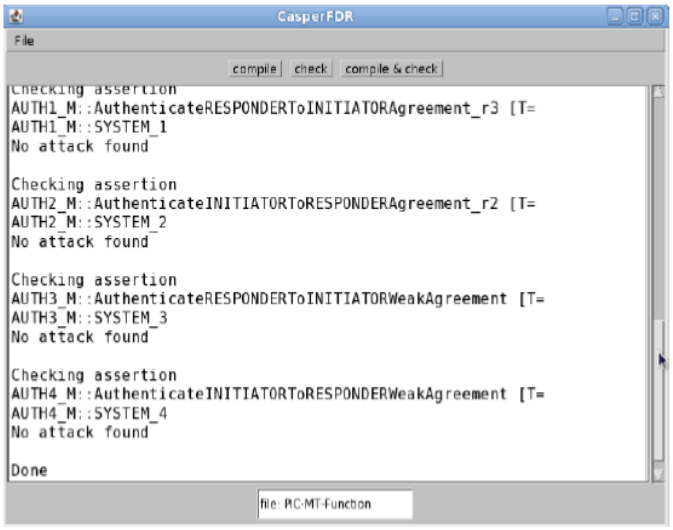

Fig. 16. Casper-FDR Verification Result

14) **EasyCrypt:** It is a cryptographic proof assistance tool related to a computational model built for simulation and game-based to formalize cryptographic proofs. Security definitions and security proofs are modeled as probabilistic and sequence conditional between different programs to represent adversaries and oracles to predefined given access [8]. Figure 17 sketches EasyCrypt workflow processes tool.
15) **Cryptoverif:** It is built on assumptions of cryptographic primitives to provide a general mechanism to deal with different types of cryptographic such as hash functions, public key PKE encryptions, message authentication codes (MAC) and symmetric encryptions in a serial sequence proofing manner [9]. It is used in many applications such as SSH, TLS 1.3, and Wired Guard Virtual private network VPN tool.
16) **F*:** Use programming language for verification of security protocol and includes different types of preconditions calculus such as monadic type, refinement type, and polymorphism to allow expressing and compact specification of programs to prove that program meet protocol specification by manual proofing and SMT solving techniques [10].

IV. Fields of Applications of the AVT

This section summarizes the AVT used in the different fields:

- Telecommunication management protocol used in management software like in ISDN, ATM, and GSM are commonly analyzed by different tools such as Inatest, NRL.
- Secure file system and trusted platform module are analyzed with Proverif tool. It can analyze authentication and security properties of protocols in different domains such as computer security domain, email protocol certifications, electronic voting systems, trusted platform modules in hardware chips, and internet key exchange in IPsec.
- AVISPA is used mostly for specifying and analyzing protocols since it has a web-based user interface that can edit protocol specifications and allow end-users to parameterize the tool's back-end configuration.
- Tamarin is used for the security protocol verification as a tool that supports both falsification and unbounded verification in the symbolic approach. Tamarin is used in many cryptographic primitives such as RSA, AES,

TABLE I
COMPARISON OF AVTS: MODELING AND SPECIFICATION

Type	AVISPA	Proverif	Coq	Tamarin	Cryptoverif	Isabelle
Programming style	imperative	functional	functional	imperative	functional	functional
Communication model	channels	channels	dependent types	channels	channels	dependent types
Cryptographic primitives	function macros	constructors	inductive types	function macros	constructors	inductive types
Protocol narration	roles	roles	set of rules	roles	roles	set of rules
Attacker model	process	–	set of rules	process	–	set of rules
Properties specification	LTL/monitor process	queries	lemmas	lemmas	queries	lemmas

TABLE II
COMPARISON OF AVTS: VERIFICATION

Type	AVISPA	Proverif	Coq	Tamarin	Cryptoverif	Isabelle
Number of sessions	bounded	unbounded	unbounded	unbounded	unbounded	bounded
Symbolic attacker	restricted	unrestricted	unrestricted	restricted	restricted	unrestricted
Message space	bounded	unbounded	unbounded	unbounded	approximated	bounded
Soundness	yes	yes	yes	yes	yes	yes
Completeness	no	no	yes	no	no	yes
Correctness of attacks	yes	maybe	yes	yes	maybe	yes
Automation	high	medium	low	high	medium	low
Guaranteed termination	yes	yes	no	yes	no	yes

and SHA3, which are also used in many application domains such as e-commerce, e-banking, and mobile communications [35]. Moreover, Tamarin has been used to analyze many authenticated key exchange protocols concerning their intended adversary models [12].

- OPT's (Origin and Path Trace) [35] security properties have been formally verified using the Coq interactive theorem prover in [36].
- Isabelle is used to give a mechanized proof of the Basic Perturbation Lemma in [37] and to verify the correctness of the Warren Abstract Machine (WAM) in [38].
- Murphy is used in signing contract protocols to verify fair exchange protocols [38] and is used in key management of 802.11i protocols and Kerberos of Telecommunication management network TMN protocols.
- Athena is used in many different protocols. It is considered an extension of the standard space model SSM used as a symbolic representation approach. It uses a reduction mechanism to eliminate the state-space problem—application domain like Kerberos authentication and authorization system.
- Brutus is used in Internet public key and comprehensive frog protocols.
- Casper is used for analyzing protocols like FDR CSP to detect errors in TMN protocols.
- CryptoVerif is used in different domains like shared and public-key encryption, signatures, MAC, and other application domains (Bluetooth, IPsec, TLS, and low power and low bandwidth protocols (Zigbee).

V. COMPARATIVE ANALYSIS OF AUTOMATED VERIFICATION TOOLS

The conducted analysis covers most of the discussed automated tools and take into account two main aspects: modelling and specification, and verification.

1) Modeling and Specification: these are essential components of the verification process. If a model does not correctly capture the semantics of a system, the results will be useless. Also, improper specification of properties may lead to erroneous results. So, it is a must to analyze each of the verification tools in terms of how precisely they can capture the semantics of cryptographic protocols as shown in Table I.
2) Verification: Table II compares the tools based on different criteria, and Table III compare different tools based on bounded verification, state space search and multiple global state.

TABLE III
TOOLS COMPARISON

Tools	Bounded Verification	State Space Search	Mutable Global State
AVISPA	Yes	Forward	Yes
Athena	Yes	Backwards	No
Casper/FDR	Yes	Forward	No
ProVerif	No	Backwards	No
StatVerif	No	Backwards	Yes
Scyther	Yes	Backwards	Yes
Tamarin	No	Backwards	Yes
Maude-NPA	No	Backwards	Yes
CryptoVerif	No	Backwards	No
EasyCrypt	No	Backwards	Yes
Scary	Yes	Forward	Yes

VI. Conclusion and Future Works:

This article presents a survey and comparative analysis of the most used automated verification tools used in cryptographic and security protocol. It describes the different types of approaches since 1983 until now.

Future work is to do a practical study to differentiate between these tools and approaches and use new programming languages to do these tools, such as Java and python. Moreover, we will choose one of the security protocols used in the 5G and RFID and do analysis and comparison using most powerful tools like AVISAP, Tamarin, Proveirf and Cryptoverif.

References

[1] S. Matsuo, K. Miyazaki, A. Otsuka, and D. Basin, How to Evaluate the Security of Real-life Cryptographic Protocols, 2007.

[2] D. Basin, M. Backes, Tools for security modeling and proofs, University of Bristol, H2020-ICT14, 2018.

[3] DANNY DOLEV AND ANDREW c. YAO, On the security of public-key protocols, IEEE Transformation Information Theory, 1983.

[4] M. Abadi and G.D. Plotkin. On protection by layout randomization. In CSF, IEEE Computer Society, 2010.

[5] M. Abadi and P. Rogaway. Reconciling two views of cryptography, the computational the soundness of formal encryption, 2007.

[6] P. Adao, C. Bozzato, G. Dei Rossi, R Focardi, and FL Luccio. A semantic-based tool for firewall configuration. Hot Issues in Security Principles and Trust, 2014.

[7] P. Agent, R. Strackx, B. Jacobs, and F. Piessens. Secure compilation to modern processors. In 25th IEEE Computer Security Foundations Symposium, 2012.

[8] G. Barthe, F. Dupressoir, B, Grfegoire, C. Kunz, B. Schmidt and P. Strub. Easycrypt: A tutorial. In Foundations of security analysis and design, 2014.

[9] B. Blanchet, A computationally sound mechanized prover for security protocols, IEEE Trans. Dependable Sec. Comput., 2008.

[10] G. Barthe, C. Fournet, B. Grfegoire, P.Strub, Nikhil Swamy, and S. Zanella Beguelin. Probabilistic relational verification for cryptographic implementations, 2014.

[11] B. Blanchet, M. Abadi, and C. Fournet, Automated verification of selected equivalences for security protocols. In Proc. 20th IEEE, 2005.

[12] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In Proc. 25th CAV, 2013.

[13] R. Sasse, S. Escobar, C. Meadows, and J. Meseguer. Protocol analysis modulo combination of theories: A case study in Maude-NPA., Security, and Trust Management, 2011.

[14] J. Ariel Hurtado Alegria, M. Cecilia Bastarrica, and A. Bergel. Analyzing software process models with Avispa, ICSSP, 2011.

[15]] S. Lortz, H. Mantel, A. Starostin, T. Bfahr, D. Schneider, and A. Weber. Cassandra: towards a certifying app store for android. In Proc. 4th ACM SPSM, 2014.

[16] Microsoft Research. Dafny: a language and program verify for functional correctness. It was accessed on 12 June 2014: http://research.Microsoft.com/en-us/projects/dafny/.

[17] Patrice Godefroid, Michael Y. Levin, and David Molnar. SAGE: White box Fuzzing for Security Testing, 2012.

[18] M. Avalle, A. Pironti, R. Sisto, Formal Verification of Security Protocols Implementations: A Survey, Informal aspects of computing, 2014.

[19] J.K. Millen, S.C. Clark, and S.B. Freedman, The Interrogator: Protocol Security Analysis, IEEE Transactions on Software Engineering, 1987.

[20] S.T. Eckmann and R.A. Kemmerer, Inatest: An Interactive Environment for Testing Formal Specifications, ACM SIGSOFT Software Engineering Notes, 1985.

[21] C. Meadows, The NRL Protocol Analyzer: An Overview, The Journal of Logic Programming, 1996.

[22] B. Blanchet, Automatic Verification of Correspondences for Security Protocols, Journal of Computer Security, 2009.

[23] C.J.F. Cremers, The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols, Proceedings of 20th International Conference Computer Aided Verification, 2008.

[24] L. Vigano, Automated Security Protocol Analysis with the AVISPA Tool 1, Electronic Notes in Theoretical Computer Science, 2006.

[25] S. Meier, B. Schmidt, C. Cremers and D.A. Basin, The TAMARIN Prover for the Symbolic Analysis of Security Protocols, Proceedings of International Conference on Computer-Aided Verification, 2013.

[26] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, and C. Murthy, "The Coq Proof Assistant Reference Manual: Version 6.1", Available at https://hal.archives-ouvertes.fr/inria-00069968/document, 2006.

[27] L.C. Paulson, "Isabelle: The Next 700 Theorem Provers", logic and computer science, 1990.

[28] J.C. Mitchell, M. Mitchell, and U. Stern, Automated Analysis of Cryptographic Protocols using Murphi, Proceedings of IEEE Symposium on Security and Privacy, 1997.

[29] D.X. Song, S. Berezin, and A. Perrig, Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis, Journal of Computer Security, 2001.

[30] F.J. Thayer, J.C. Herzog, and J.D. Guttman, Strand Spaces: Why is a Security Protocol Correct Proceedings of IEEE Symposium on Security and Privacy, 1998.

[31] E.M. Clarke, S. Jha, and W.R. Marrero Verifying Security Protocols with Brutus, ACM Transactions on Software Engineering and Methodology, 2000.

[32] G. Lowe, "Casper: A Compiler for the Analysis of Security Protocols," Journal of Computer Security, 1998.

[33] A. W. Roscoe, Model-Checking CSP, Prentice, 1994.

[34] A. Armando, D. Basin, Y. Boichut, The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications, Conference Paper in Lecture Notes in Computer Science, 2005.

[35] F. Zhang, L. Jia, C. Basescu, T.H. Kim, Y. Hu, and A. Perrig, Mechanized Network Origin and Path Authenticity Proofs, Proceedings of Conference on Computer and Communications Security, 2014.

[36] J. Aransay, C. Ballarin, and J. Rubio, A Mechanized Proof of the Basic Perturbation Lemma, Journal of Automated Reasoning, 2008.

[37] C. Pusch, Verification of Compiler Correctness for the WAM, Proceedings of International Conference on Theorem Proving in Higher Order Logics, 1996.

[38] H. Miguel PalomboA Comparative Study of Formal Verification Techniques for Authentication Protocols, Department of Computer Science and Engineering College of Engineering University of South Florida, 2015.