I. J. Computer Network and Information Security, **2016, 3, 34-40**
Published Online March 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2016.03.05

Analysis of Cryptographic Protocols AKI, ARPKI and OPT using ProVerif and AVISPA

Amol H. Shinde
Department of Information Technology, WCE, Sangli, Maharashtra, India.
E-mail: shinde.amol1991@gmail.com

A. J. Umbarkar
Department of Information Technology, WCE, Sangli, Maharashtra, India.
E-mail: anantumbarkar@rediffmail.com

Abstract—In recent years, the area of formal verification of cryptographic protocols became important because of the active intruders. These intruders can find out the flaws in the protocols and can use them to create attacks. To avoid such possible attacks, the protocols must be verified to check if the protocols contain any flaws. The formal verification tools have helped in verifying and correcting the protocols. Various tools are available these days for verifying the protocols. In this paper, the two verification tools namely ProVerif and AVISPA are used for analysis of protocols - AKI (Accountable Key Infrastructure), ARPKI (Attack Resilient Public Key Infrastructure) and OPT (Origin and Path Trace). A comparative evaluation of the selected tools is presented and revealed security properties of the protocols selected.

Index Terms—Formal Verification, Cryptographic Protocols, ProVerif, AVISPA, Comparison of Tools.

I. Introduction

A communication protocol which uses the cryptographic operations is known as the Cryptographic protocol. Cryptographic protocol contains functions such as distribution of keys to the entities, authentication of principals to each other to make the secure transaction or computation over the network. The network is usually assumed to be hostile. The network may contain adversaries who are able to read modify and delete traffic and may have control over some of the principals working in the network. The adversary is able to manipulate the data used by the protocol with which the adversary can form the attacks on the protocol. Some attacks may be dependent on the subtle properties of the cryptographic algorithms or the statistical analysis of message traffic. Formal methods are used to check if such attacks are possible on the protocol.

Formal methods are a combination of a mathematical or a logical model of a system and its requirements, together with an effective procedure for determining whether the proof that a system satisfies its requirements is correct [1]. The use of formal methods for verification has caused developing the tools to verify cryptographic protocols. Some of the verification tools developed are ProVerif [2], AVISPA [3], Scyther [4], TAMARIN [5], Athena [6], NRL protocol analyzer [7], to name a few. The tools differ in their input language, the way of verification and the way in which the output is provided.

In this paper, an attempt is made to evaluate the two popular cryptographic verification tools namely ProVerif and AVISPA. Newly proposed cryptographic protocols AKI (Accountable Key Infrastructure) [8], ARPKI (Attack Resilient Public Key Infrastructure) [9] and OPT (Origin and Path Trace) [10] are analyzed using both these tools.

Paper is organized as follows: Section II describes the literature survey which includes a brief introduction of ProVerif and AVISPA tools. In Section III, details of selected protocols are presented. Section IV discusses the comparative analysis of the two tools and in Section V the conclusion and probable future work is discussed.

II. Literature Survey

There are various tools present for the verification of the cryptographic protocols such as Interrogator [11], NRL protocol analyzer [7], ProVerif [2], Scyther [4], AVISPA [3], Isabelle [12], TAMARIN [5], Coq theorem prover [13], Athena [6], Brutus [14], Murφ [15] etc. In this work, AVISPA and ProVerif are used for the comparative evaluation. The characteristics of these two tools are as follows.

A. ProVerif

ProVerif is a tool for automatically analyzing the security of cryptographic protocols [2]. This is developed by Bruno Blanchet. This tool verifies the protocol for an unbounded number of sessions, using unbounded message space. ProVerif provides an automatic technique to verify correspondences in the protocols. The user has to code the protocol and the correspondences. Correspondences are the properties of the form; if the protocol executes some event then the protocol must have executed some other events before. The events can be described using a logical formula which can contain conjunctions and disjunctions. The tool is capable of attack reconstruction. If a property cannot be proved, the trace which falsifies the desired property is built. In two ways the input can be provided to this tool, Horn clauses or Pi calculus. Attacker is not specified explicitly. The tool is very precise even if with the limitation that in rare cases the solving algorithm does not terminate.

B. AVISPA

AVISPA [3] is a push-button tool for the Automated Validation of Internet Security Protocols and Applications. An expressive formal language named HLPSL (High Level Protocol Specification Language) is provided by AVISPA for specifying protocols and properties. AVISPA combines different back-ends having a variety of automatic protocol analysis techniques ranging from protocol falsification to abstraction based verification methods for infinite number of sessions. The current version of AVISPA integrates four back ends viz. the On-the-fly-model-checker (OFMC), the Constraint-Logic-based Attack Searcher (CL-AtSe), the SAT-based Model-Checker (SATMC) and the Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP) protocol Analyzer [3]. These back-ends analyze the protocols assuming there is perfect cryptography and the messages exchanged over the network are under the control of Dolev-Yao intruder. When protocol terminates, each back end outputs the result of analysis using output format stating whether the input format is solved, system resources are exhausted or the problem is not tackled for some reason.

III. PROTOCOLS

In order to analyze the tools, three cryptographic protocols are identified, implemented and analyzed using both ProVerif and AVISPA. The protocols selected are AKI [8], ARPKI [9] and OPT [10]. Researchers have proposed these new protocols, which will possibly be used in the next generation of network. As these protocols are likely to be used in the future; these protocols must be secure to operate in the hostile environment. In rest of this section the selected protocols are discussed in brief.

A. AKI

Accountable Key Infrastructure (AKI) [8] is a new public key infrastructure. AKI is used to reduce the levels of trust in Certification Authorities (CAs) as research says that now-a-days there is decreased trust in CAs. AKI decentralizes the certification authority and distributes that work between different entities. All these entities monitor each other's work to avoid any malicious activity. AKI combines an accountability infrastructure i.e. providing checks and balances on server operations and misbehavior dissemination, with key revocation mechanisms.

The AKI contains following entities.

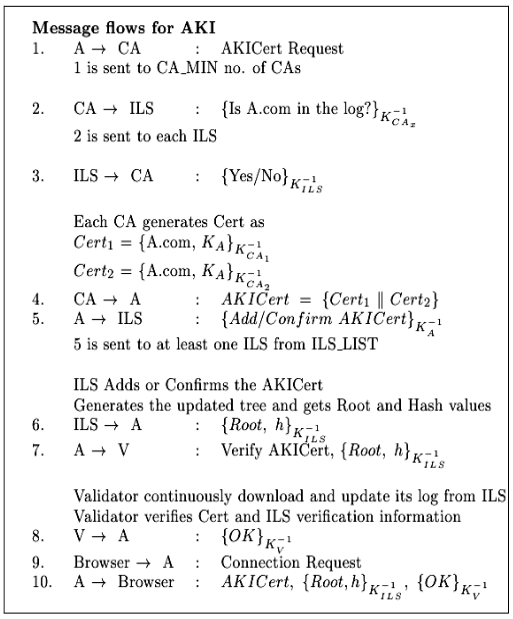
- A Domain (server) is a named entity with which clients desire to establish secure connections.
- A Client (browser) is an entity establishing TLS connections with domains (servers).
- A Certification Agency is similar to current certification authority which authenticates domains and issues X.509 certificates.
- An Integrity Log Server (ILS) keeps an Integrity Tree that stores CA-issued certificates to make them publicly visible.

This Integrity Tree is a hash tree of all the registered certificates in lexicographic order. Each ILS updates its Integrity Tree at a given interval, known as ILS_UP.

- Validators monitor ILS operations to detect misbehavior like sudden (dis)appearance of certificates.

Alice owns a domain A.com and she wants to obtain an AKI-protected certificate. She defines CAs and ILSs that she trusts (CA_LIST and ILS_LIST respectively), the minimum number of CA signatures that she recommends her client for validation (CA_MIN) and rules for certificate revocation, replacement and updates. Alice with her public key, contacts more than the minimum number of trusted CAs to sign her certificate. Alice registers the certificate with one or multiple ILSs after the signing of the certificate. Each ILS then adds A.com to its database by placing it into the Integrity Tree. ILS then re-computes hash values and updates the tree for updated verification information.

Whenever browsers connect to Alice's website via HTTPS, Alice supplements her certificate with the verification information that she downloads from every ILS and sends it to browsers. Browser uses the pre-installed ILS public key(s) on her browser to validate ILS information. The client browser occasionally checks with validator, who is continuously updating data by downloading entire ILS data, to confirm that the ILSs' current root hash values are valid. The message flows for AKI is shown in Fig. 1.


Message flows for AKI

1. A $\rightarrow$ CA : AKICert Request
   1 is sent to CA_MIN no. of CAs

2. CA $\rightarrow$ ILS : $\{\text{Is A.com in the log?}\}_{K^{-1}_{CA_x}}$
   2 is sent to each ILS

3. ILS $\rightarrow$ CA : $\{\text{Yes/No}\}_{K^{-1}_{ILS}}$

   Each CA generates Cert as
   $Cert_1 = \{\text{A.com}, K_A\}_{K^{-1}_{CA_1}}$
   $Cert_2 = \{\text{A.com}, K_A\}_{K^{-1}_{CA_2}}$

4. CA $\rightarrow$ A : $AKICert = \{Cert_1 \parallel Cert_2\}$
5. A $\rightarrow$ ILS : $\{Add/Confirm\ AKICert\}_{K^{-1}_A}$
   5 is sent to at least one ILS from ILS_LIST

   ILS Adds or Confirms the AKICert
   Generates the updated tree and gets Root and Hash values

6. ILS $\rightarrow$ A : $\{Root,\ h\}_{K^{-1}_{ILS}}$
7. A $\rightarrow$ V : Verify AKICert, $\{Root,\ h\}_{K^{-1}_{ILS}}$

   Validator continuously download and update its log from ILS
   Validator verifies Cert and ILS verification information

8. V $\rightarrow$ A : $\{OK\}_{K^{-1}_V}$
9. Browser $\rightarrow$ A : Connection Request
10. A $\rightarrow$ Browser : $AKICert$, $\{Root, h\}_{K^{-1}_{ILS}}$, $\{OK\}_{K^{-1}_V}$

Fig.1. Message Flows for AKI

B. ARPKI

Attack Resilient Public-Key Infrastructure (ARPKI) [9] is also a new public key infrastructure. ARPKI ensures that certificate related operations are transparent and accountable. ARPKI is very closely related to AKI. ARPKI is inspired by AKI's design and uses some of its concepts. In ARPKI there are also three entities as there in AKI, these entities work to authenticate a new certificate to domain, confirm and update that certificate. ARPKI offers extremely strong security guarantees, where compromising n-1 out of n trusted signing and verifying entities is insufficient to launch an impersonation attack [9].

ARPKI uses three entities for the certificate operations which are two CAs (Certification Agencies) and one ILS (Integrity Log Server). ARPKI's CAs conduct active on-line confirmations with validator-like capabilities.

Summary of actors and their responsibilities in ARPKI:

- A domain registers ARPKI certificate (ARCert) for itself with ARPKI infrastructure and can use it for securely serving the web pages to the clients.
- The CAs check the identity of the domain owner on registration and then sign and give guarantees for the presented certificate.
- The CAs are responsible for checking the logs for this ARCert and assuring the correct operation of other entities involved in creating the ARCert.
- The CAs download all accepted requests from the ILSes and compare them to the published integrity trees, to check the ILSes behavior.
- The ILSes keep a log of all ARCerts registered with them, and provide proofs of existence for ARCerts that are then used by CAs and domains.
- Optionally there can be additional validators that execute checks similar to those made by CAs, but without issuing ARCerts themselves.

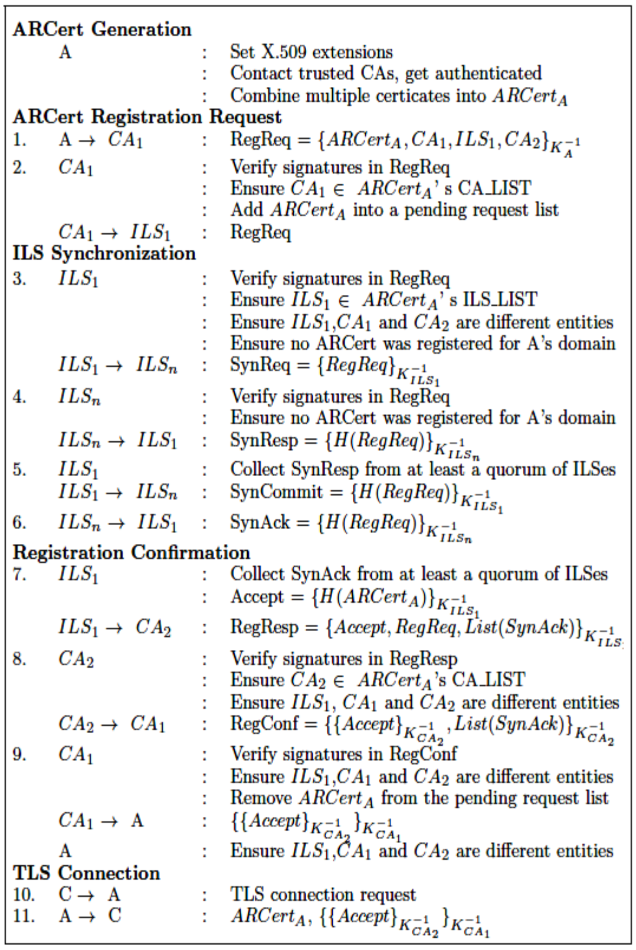
There are three parts of protocol, ARCert generation, Confirmation and certificate updation. Fig. 2, Fig. 3 and Fig. 4 show the message flow for each protocol respectively.

C. OPT

Origin and Path Trace (OPT) [10] is lightweight, scalable, and secure protocol for source authentication and path validation. Source authentication means the destination and each intermediate router should be able to determine whether the packet indeed originated from the claimed source and whether the packet content has not been altered en route [10]. Path validation means the source, intermediate routers, and the destination should be able to validate that the packet indeed traversed the path known to (or selected by) the source [10].

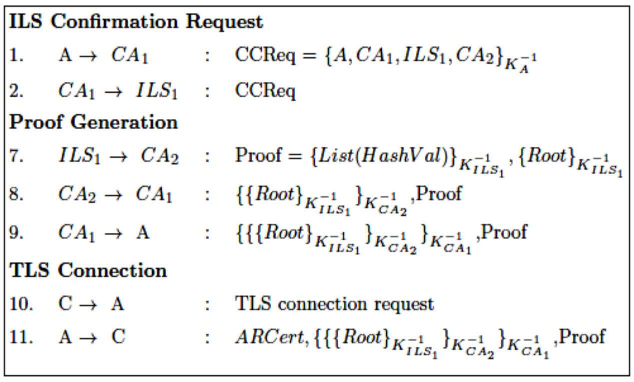
Let S be source and D be destination assure that S sends the packets to D along the sequence of routers Ri. In the packet header, source S includes H(P), which is the hash of the packet payload. This helps receiving entities to identify the packet. Each router Ri, on demand, generates key Ki using a symmetric cryptographic operation. OPT uses the keys generated by DRKey protocol. DRKey (Dynamically Re-creatable key) protocol is used for distributing the shared keys between the participant entities. DRKey protocol is of two types, one is when source and destination trust each other and other is when source and destination do not trust each other. Fig. 5 and Fig. 6 show these two DRKey protocols respectively.

DRKey protocols generate the symmetric keys which will be useful for carrying out OPT protocol. The message flow for the OPT protocol is shown in Fig. 7.



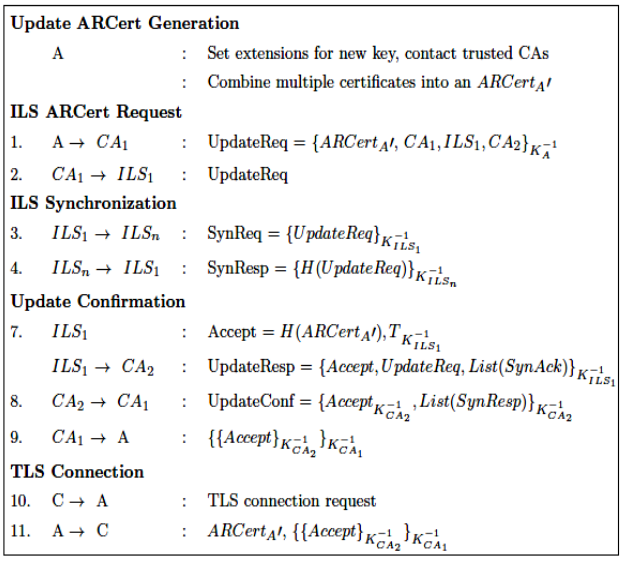
ARCert Generation			
	A	:	Set X.509 extensions
		:	Contact trusted CAs, get authenticated
		:	Combine multiple certicates into $ARCert_A$
ARCert Registration Request			
1.	A $\rightarrow$ CA_1	:	RegReq = $\{ARCert_A, CA_1, ILS_1, CA_2\}_{K_A^{-1}}$
2.	CA_1	:	Verify signatures in RegReq
		:	Ensure $CA_1 \in$ $ARCert_A$' s CA_LIST
		:	Add $ARCert_A$ into a pending request list
	$CA_1 \rightarrow ILS_1$	:	RegReq
ILS Synchronization			
3.	ILS_1	:	Verify signatures in RegReq
		:	Ensure $ILS_1 \in$ $ARCert_A$' s ILS_LIST
		:	Ensure ILS_1,CA_1 and CA_2 are different entities
		:	Ensure no ARCert was registered for A's domain
	$ILS_1 \rightarrow ILS_n$	:	SynReq = $\{RegReq\}_{K_{ILS_1}^{-1}}$
4.	ILS_n	:	Verify signatures in RegReq
		:	Ensure no ARCert was registered for A's domain
	$ILS_n \rightarrow ILS_1$	:	SynResp = $\{H(RegReq)\}_{K_{ILS_n}^{-1}}$
5.	ILS_1	:	Collect SynResp from at least a quorum of ILSes
	$ILS_1 \rightarrow ILS_n$	:	SynCommit = $\{H(RegReq)\}_{K_{ILS_1}^{-1}}$
6.	$ILS_n \rightarrow ILS_1$	:	SynAck = $\{H(RegReq)\}_{K_{ILS_n}^{-1}}$
Registration Confirmation			
7.	ILS_1	:	Collect SynAck from at least a quorum of ILSes
		:	Accept = $\{H(ARCert_A)\}_{K_{ILS_1}^{-1}}$
	$ILS_1 \rightarrow CA_2$	:	RegResp = $\{Accept, RegReq, List(SynAck)\}_{K_{ILS}^{-1}}$
8.	CA_2	:	Verify signatures in RegResp
		:	Ensure $CA_2 \in$ $ARCert_A$'s CA_LIST
		:	Ensure ILS_1, CA_1 and CA_2 are different entities
	$CA_2 \rightarrow CA_1$	:	RegConf = $\{\{Accept\}_{K_{CA_2}^{-1}}, List(SynAck)\}_{K_{CA_2}^{-1}}$
9.	CA_1	:	Verify signatures in RegConf
		:	Ensure ILS_1,CA_1 and CA_2 are different entities
		:	Remove $ARCert_A$ from the pending request list
	$CA_1 \rightarrow$ A	:	$\{\{Accept\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$
	A	:	Ensure ILS_1,CA_1 and CA_2 are different entities
TLS Connection			
10.	C $\rightarrow$ A	:	TLS connection request
11.	A $\rightarrow$ C	:	$ARCert_A$, $\{\{Accept\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$

Fig.2. ARcert Generation



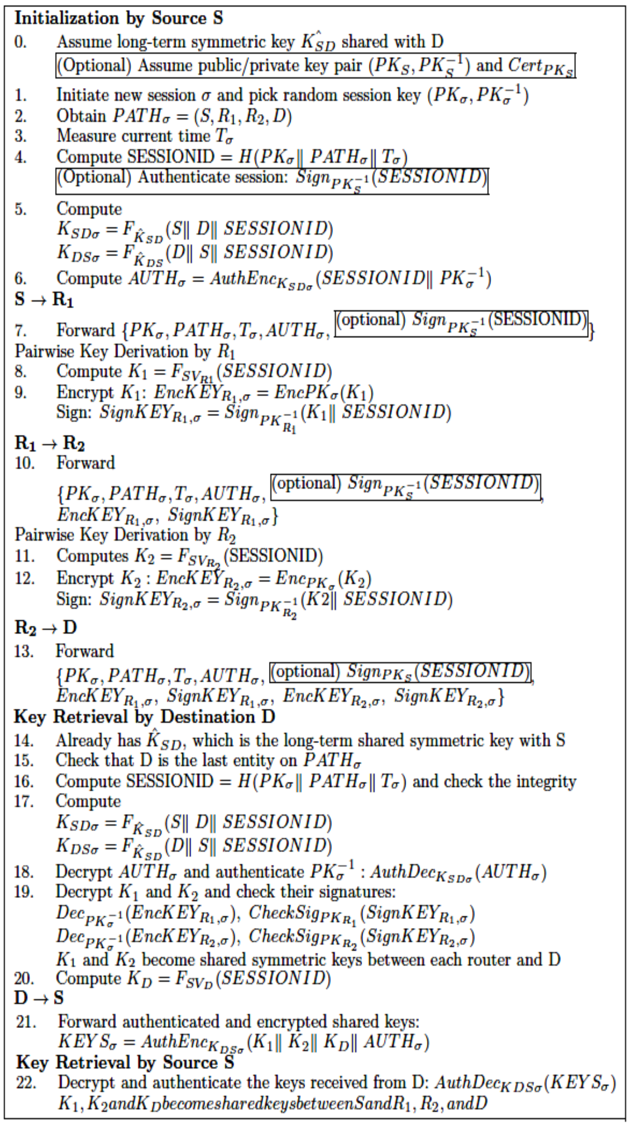
ILS Confirmation Request			
1.	A $\rightarrow$ CA_1	:	CCReq = $\{A, CA_1, ILS_1, CA_2\}_{K_A^{-1}}$
2.	$CA_1 \rightarrow ILS_1$	:	CCReq
Proof Generation			
7.	$ILS_1 \rightarrow CA_2$	:	Proof = $\{List(HashVal)\}_{K_{ILS_1}^{-1}}, \{Root\}_{K_{ILS_1}^{-1}}$
8.	$CA_2 \rightarrow CA_1$	:	$\{\{Root\}_{K_{ILS_1}^{-1}}\}_{K_{CA_2}^{-1}}$,Proof
9.	$CA_1 \rightarrow$ A	:	$\{\{\{Root\}_{K_{ILS_1}^{-1}}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$,Proof
TLS Connection			
10.	C $\rightarrow$ A	:	TLS connection request
11.	A $\rightarrow$ C	:	$ARCert, \{\{\{Root\}_{K_{ILS_1}^{-1}}\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$,Proof

Fig.3. ARCert Confirmation



Update ARCert Generation			
	A	:	Set extensions for new key, contact trusted CAs
		:	Combine multiple certificates into an $ARCert_{A'}$
ILS ARCert Request			
1.	$A \rightarrow CA_1$	:	UpdateReq $= \{ARCert_{A'}, CA_1, ILS_1, CA_2\}_{K_A^{-1}}$
2.	$CA_1 \rightarrow ILS_1$	:	UpdateReq
ILS Synchronization			
3.	$ILS_1 \rightarrow ILS_n$	:	SynReq $= \{UpdateReq\}_{K_{ILS_1}^{-1}}$
4.	$ILS_n \rightarrow ILS_1$	:	SynResp $= \{H(UpdateReq)\}_{K_{ILS_n}^{-1}}$
Update Confirmation			
7.	ILS_1	:	Accept $= H(ARCert_{A'}), T_{K_{ILS_1}^{-1}}$
	$ILS_1 \rightarrow CA_2$	:	UpdateResp $= \{Accept, UpdateReq, List(SynAck)\}_{K_{ILS_1}^{-1}}$
8.	$CA_2 \rightarrow CA_1$	:	UpdateConf $= \{Accept_{K_{CA_2}^{-1}}, List(SynResp)\}_{K_{CA_2}^{-1}}$
9.	$CA_1 \rightarrow A$	:	$\{\{Accept\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$
TLS Connection			
10.	$C \rightarrow A$	:	TLS connection request
11.	$A \rightarrow C$	:	$ARCert_{A'}, \{\{Accept\}_{K_{CA_2}^{-1}}\}_{K_{CA_1}^{-1}}$

Fig.4. ARCert Updation



Initialization by Source S

0. Assume long-term symmetric key $\hat{K_{SD}}$ shared with D
 $\boxed{\text{(Optional) Assume public/private key pair } (PK_S, PK_S^{-1}) \text{ and } Cert_{PK_S}}$
1. Initiate new session σ and pick random session key $(PK_\sigma, PK_\sigma^{-1})$
2. Obtain $PATH_\sigma = (S, R_1, R_2, D)$
3. Measure current time T_σ
4. Compute SESSIONID $= H(PK_\sigma \| PATH_\sigma \| T_\sigma)$
 $\boxed{\text{(Optional) Authenticate session: } Sign_{PK_S^{-1}}(SESSIONID)}$
5. Compute
 $K_{SD\sigma} = F_{\hat{K}_{SD}}(S \| D \| SESSIONID)$
 $K_{DS\sigma} = F_{\hat{K}_{DS}}(D \| S \| SESSIONID)$
6. Compute $AUTH_\sigma = AuthEnc_{K_{SD\sigma}}(SESSIONID \| PK_\sigma^{-1})$

S → R₁

7. Forward $\{PK_\sigma, PATH_\sigma, T_\sigma, AUTH_\sigma, \boxed{\text{(optional) } Sign_{PK_S^{-1}}(\text{SESSIONID})}\}$

Pairwise Key Derivation by R_1

8. Compute $K_1 = F_{SV_{R1}}(SESSIONID)$
9. Encrypt K_1: $EncKEY_{R_1,\sigma} = EncPK_\sigma(K_1)$
 Sign: $SignKEY_{R_1,\sigma} = Sign_{PK_{R_1}^{-1}}(K_1 \| SESSIONID)$

R₁ → R₂

10. Forward
 $\{PK_\sigma, PATH_\sigma, T_\sigma, AUTH_\sigma, \boxed{\text{(optional) } Sign_{PK_S^{-1}}(SESSIONID)},$
 $EncKEY_{R_1,\sigma},\ SignKEY_{R_1,\sigma}\}$

Pairwise Key Derivation by R_2

11. Computes $K_2 = F_{SV_{R_2}}$(SESSIONID)
12. Encrypt $K_2 : EncKEY_{R_2,\sigma} = Enc_{PK_\sigma}(K_2)$
 Sign: $SignKEY_{R_2,\sigma} = Sign_{PK_{R_2}^{-1}}(K2 \| SESSIONID)$

R₂ → D

13. Forward
 $\{PK_\sigma, PATH_\sigma, T_\sigma, AUTH_\sigma, \boxed{\text{(optional) } Sign_{PK_S}(SESSIONID)},$
 $EncKEY_{R_1,\sigma},\ SignKEY_{R_1,\sigma},\ EncKEY_{R_2,\sigma},\ SignKEY_{R_2,\sigma}\}$

Key Retrieval by Destination D

14. Already has $\hat{K}_{SD}$, which is the long-term shared symmetric key with S
15. Check that D is the last entity on $PATH_\sigma$
16. Compute SESSIONID $= H(PK_\sigma \| PATH_\sigma \| T_\sigma)$ and check the integrity
17. Compute
 $K_{SD\sigma} = F_{\hat{K}_{SD}}(S \| D \| SESSIONID)$
 $K_{DS\sigma} = F_{\hat{K}_{SD}}(D \| S \| SESSIONID)$
18. Decrypt $AUTH_\sigma$ and authenticate PK_σ^{-1} : $AuthDec_{K_{SD\sigma}}(AUTH_\sigma)$
19. Decrypt K_1 and K_2 and check their signatures:
 $Dec_{PK_\sigma^{-1}}(EncKEY_{R_1,\sigma}),\ CheckSig_{PK_{R_1}}(SignKEY_{R_1,\sigma})$
 $Dec_{PK_\sigma^{-1}}(EncKEY_{R_2,\sigma}),\ CheckSig_{PK_{R_2}}(SignKEY_{R_2,\sigma})$
 K_1 and K_2 become shared symmetric keys between each router and D
20. Compute $K_D = F_{SV_D}(SESSIONID)$

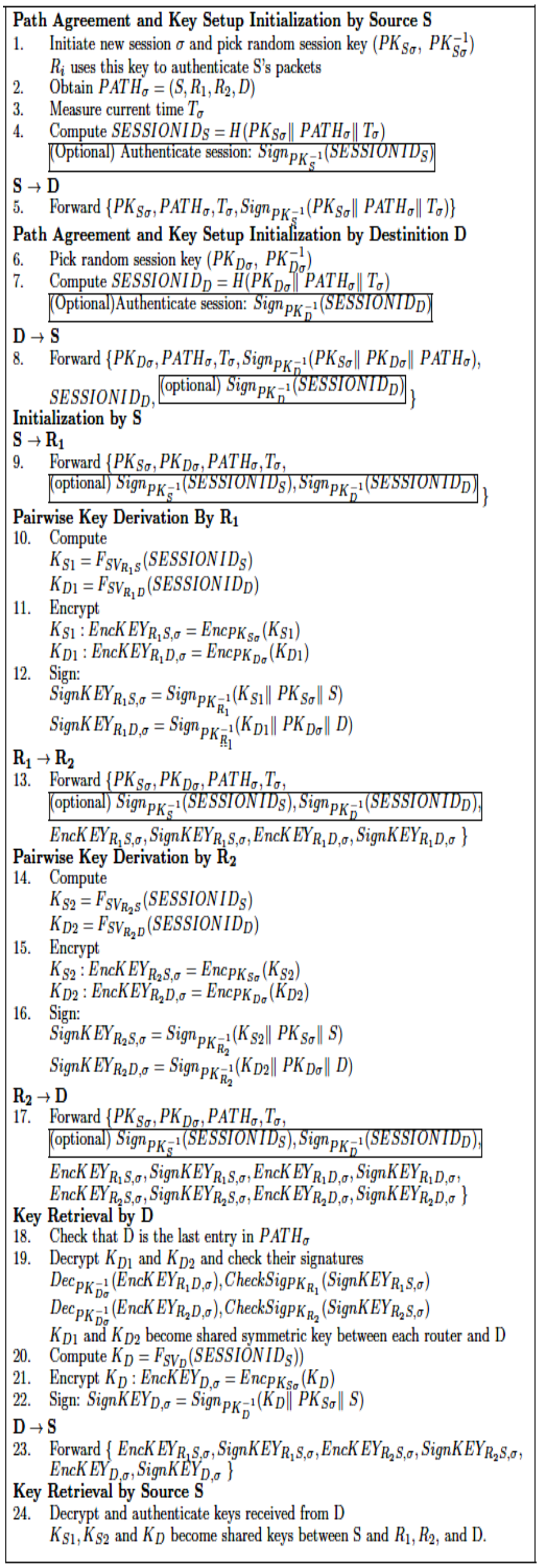
D → S

21. Forward authenticated and encrypted shared keys:
 $KEYS_\sigma = AuthEnc_{K_{DS\sigma}}(K_1 \| K_2 \| K_D \| AUTH_\sigma)$

Key Retrieval by Source S

22. Decrypt and authenticate the keys received from D: $AuthDec_{KDS\sigma}(KEYS_\sigma)$
 $K_1, K_2 and K_D becomesharedkeysbetweenSandR_1, R_2, andD$

Fig.5. DRKey Protocol When S and D Trust Each Other



Path Agreement and Key Setup Initialization by Source S

1. Initiate new session σ and pick random session key $(PK_{S\sigma},\ PK_{S\sigma}^{-1})$
 R_i uses this key to authenticate S's packets
2. Obtain $PATH_\sigma = (S, R_1, R_2, D)$
3. Measure current time T_σ
4. Compute $SESSIONID_S = H(PK_{S\sigma} \| PATH_\sigma \| T_\sigma)$
 $\boxed{\text{(Optional) Authenticate session: } Sign_{PK_S^{-1}}(SESSIONID_S)}$

S → D

5. Forward $\{PK_{S\sigma}, PATH_\sigma, T_\sigma, Sign_{PK_S^{-1}}(PK_{S\sigma} \| PATH_\sigma \| T_\sigma)\}$

Path Agreement and Key Setup Initialization by Destinition D

6. Pick random session key $(PK_{D\sigma},\ PK_{D\sigma}^{-1})$
7. Compute $SESSIONID_D = H(PK_{D\sigma} \| PATH_\sigma \| T_\sigma)$
 $\boxed{\text{(Optional)Authenticate session: } Sign_{PK_D^{-1}}(SESSIONID_D)}$

D → S

8. Forward $\{PK_{D\sigma}, PATH_\sigma, T_\sigma, Sign_{PK_D^{-1}}(PK_{S\sigma} \| PK_{D\sigma} \| PATH_\sigma),$
 $SESSIONID_D, \boxed{\text{(optional) } Sign_{PK_D^{-1}}(SESSIONID_D)}\}$

Initialization by S

S → R₁

9. Forward $\{PK_{S\sigma}, PK_{D\sigma}, PATH_\sigma, T_\sigma,$
 $\boxed{\text{(optional) } Sign_{PK_S^{-1}}(SESSIONID_S), Sign_{PK_D^{-1}}(SESSIONID_D)}\}$

Pairwise Key Derivation By R₁

10. Compute
 $K_{S1} = F_{SV_{R_1S}}(SESSIONID_S)$
 $K_{D1} = F_{SV_{R_1D}}(SESSIONID_D)$
11. Encrypt
 $K_{S1} : EncKEY_{R_1S,\sigma} = Enc_{PK_{S\sigma}}(K_{S1})$
 $K_{D1} : EncKEY_{R_1D,\sigma} = Enc_{PK_{D\sigma}}(K_{D1})$
12. Sign:
 $SignKEY_{R_1S,\sigma} = Sign_{PK_{R_1}^{-1}}(K_{S1} \| PK_{S\sigma} \| S)$
 $SignKEY_{R_1D,\sigma} = Sign_{PK_{R_1}^{-1}}(K_{D1} \| PK_{D\sigma} \| D)$

R₁ → R₂

13. Forward $\{PK_{S\sigma}, PK_{D\sigma}, PATH_\sigma, T_\sigma,$
 $\boxed{\text{(optional) } Sign_{PK_S^{-1}}(SESSIONID_S), Sign_{PK_D^{-1}}(SESSIONID_D),}$
 $EncKEY_{R_1S,\sigma}, SignKEY_{R_1S,\sigma}, EncKEY_{R_1D,\sigma}, SignKEY_{R_1D,\sigma}\}$

Pairwise Key Derivation by R₂

14. Compute
 $K_{S2} = F_{SV_{R_2S}}(SESSIONID_S)$
 $K_{D2} = F_{SV_{R_2D}}(SESSIONID_D)$
15. Encrypt
 $K_{S2} : EncKEY_{R_2S,\sigma} = Enc_{PK_{S\sigma}}(K_{S2})$
 $K_{D2} : EncKEY_{R_2D,\sigma} = Enc_{PK_{D\sigma}}(K_{D2})$
16. Sign:
 $SignKEY_{R_2S,\sigma} = Sign_{PK_{R_2}^{-1}}(K_{S2} \| PK_{S\sigma} \| S)$
 $SignKEY_{R_2D,\sigma} = Sign_{PK_{R_2}^{-1}}(K_{D2} \| PK_{D\sigma} \| D)$

R₂ → D

17. Forward $\{PK_{S\sigma}, PK_{D\sigma}, PATH_\sigma, T_\sigma,$
 $\boxed{\text{(optional) } Sign_{PK_S^{-1}}(SESSIONID_S), Sign_{PK_D^{-1}}(SESSIONID_D),}$
 $EncKEY_{R_1S,\sigma}, SignKEY_{R_1S,\sigma}, EncKEY_{R_1D,\sigma}, SignKEY_{R_1D,\sigma},$
 $EncKEY_{R_2S,\sigma}, SignKEY_{R_2S,\sigma}, EncKEY_{R_2D,\sigma}, SignKEY_{R_2D,\sigma}\}$

Key Retrieval by D

18. Check that D is the last entry in $PATH_\sigma$
19. Decrypt K_{D1} and K_{D2} and check their signatures
 $Dec_{PK_{D\sigma}^{-1}}(EncKEY_{R_1D,\sigma}), CheckSig_{PK_{R_1}}(SignKEY_{R_1S,\sigma})$
 $Dec_{PK_{D\sigma}^{-1}}(EncKEY_{R_2D,\sigma}), CheckSig_{PK_{R_2}}(SignKEY_{R_2S,\sigma})$
 K_{D1} and K_{D2} become shared symmetric key between each router and D
20. Compute $K_D = F_{SV_D}(SESSIONID_S))$
21. Encrypt $K_D : EncKEY_{D,\sigma} = Enc_{PK_{S\sigma}}(K_D)$
22. Sign: $SignKEY_{D,\sigma} = Sign_{PK_D^{-1}}(K_D \| PK_{S\sigma} \| S)$

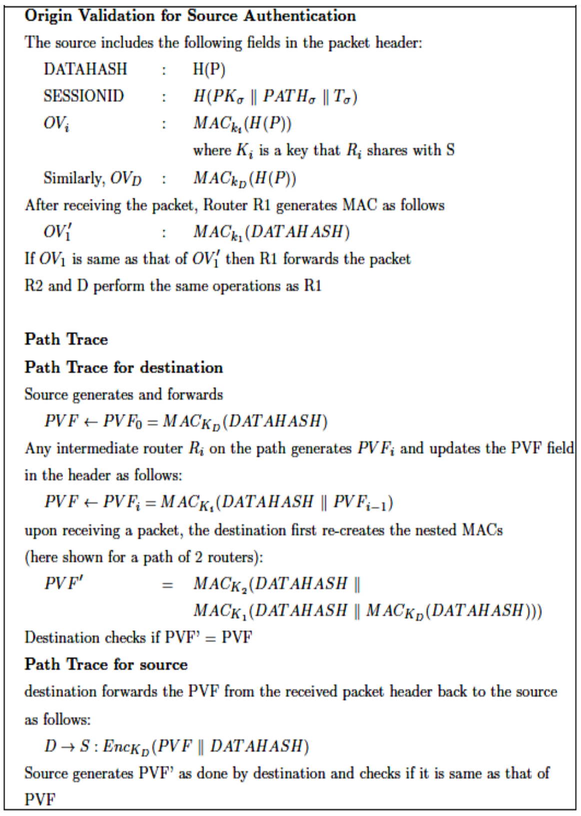
D → S

23. Forward $\{EncKEY_{R_1S,\sigma}, SignKEY_{R_1S,\sigma}, EncKEY_{R_2S,\sigma}, SignKEY_{R_2S,\sigma},$
 $EncKEY_{D,\sigma}, SignKEY_{D,\sigma}\}$

Key Retrieval by Source S

24. Decrypt and authenticate keys received from D
 K_{S1}, K_{S2} and K_D become shared keys between S and R_1, R_2, and D.

Fig.6. DRKey Protocol When S and D Do Not Trust Each Other



Origin Validation for Source Authentication

The source includes the following fields in the packet header:

DATAHASH : H(P)

SESSIONID : $H(PK_\sigma \parallel PATH_\sigma \parallel T_\sigma)$

OV_i : $MAC_{k_i}(H(P))$

where K_i is a key that R_i shares with S

Similarly, OV_D : $MAC_{k_D}(H(P))$

After receiving the packet, Router R1 generates MAC as follows

OV_1' : $MAC_{k_1}(DATAHASH)$

If OV_1 is same as that of OV_1' then R1 forwards the packet

R2 and D perform the same operations as R1

Path Trace

Path Trace for destination

Source generates and forwards

$PVF \leftarrow PVF_0 = MAC_{K_D}(DATAHASH)$

Any intermediate router R_i on the path generates PVF_i and updates the PVF field in the header as follows:

$PVF \leftarrow PVF_i = MAC_{K_i}(DATAHASH \parallel PVF_{i-1})$

upon receiving a packet, the destination first re-creates the nested MACs (here shown for a path of 2 routers):

$$PVF' = MAC_{K_2}(DATAHASH \parallel MAC_{K_1}(DATAHASH \parallel MAC_{K_D}(DATAHASH)))$$

Destination checks if PVF' = PVF

Path Trace for source

destination forwards the PVF from the received packet header back to the source as follows:

$D \rightarrow S : Enc_{K_D}(PVF \parallel DATAHASH)$

Source generates PVF' as done by destination and checks if it is same as that of PVF

Fig.7. OPT Protocol

IV. Analysis Results

The verification of the selected protocols is done by using ProVerif and AVISPA. On-the-fly Model-Checker (OFMC) is one of the back-ends of AVISPA which is used for the analysis of the selected protocols. OFMC performs both protocol falsification and bounded session verification, by exploring the transition system described by an IF specification in a demand-driven way (i.e., on-the-fly) [3]. The specifications of the protocols are as per our interpretation of the protocols. The analysis results of both the tools for each protocol are discussed below.

A. Analysis of AKI

- Analysis using AVISPA

Analysis of the protocol AKI using AVISPA (OFMC) gave an attack trace in which it showed that the intruder can get the certificate for any domain. The CA should check that the message sent by the domain should contain its own information. So that CA will not give the certificate for intruder who is requesting it for another domain. In attack trace it is found that intruder can get the certificate for domain A by sending the public information of A.

Secrecy of the certificate contents was not an issue, integrity is, but that is being assured to be taken care of by other means.

- Analysis using ProVerif

In the analysis of AKI, it is found that the injective correspondences are false but some related non-injective correspondences are true.

Correspondence query is true if and only if, for all executions of the protocol, if the event to the left of the arrow has been executed, then the event to the right of the arrow has also been executed before. Injective correspondence asserts that, for each occurrence of the event to the left of the arrow, there is a distinct earlier occurrence of the event to the right of the arrow.

The false correspondence query shows that the event specified to the left of the arrow can occur without any previous occurrence of the event specified to the right of the arrow, which means authentication failure.

In the result, it is found that the authentication of CA to A violated same was violated in result of AVISPA. Authentication of ILS to A and authentication of A to ILS are also violated but the non injective authentication queries are true for them.

B. Analysis of ARPKI

- Analysis using AVISPA

In registration process, the intruder can get the certificate for the domain if the CA1 does not check that the domain is registering its own certificate. The intruder can send the public data of domain which is necessary to get the certificate and if CA1 does not check the data, then the intruder can get the certificate for the domain for a public key of intruder's choice.

A sends message (ARCert, CA1, CA2, ILS1) encrypted with private key of A to CA1 to register the certificate. CA1 sends this message to ILS1 and CA2 to register the certificate, after registering, CA1 will get Accept message from CA2, which it will send it to A. In updation process also this situation is possible, that the intruder can get the certificate from the CA1 by giving the information of the domain.

In confirmation process, it is found that the authentication of the Root value fails. The Root value is the root of the hash tree known as integrity tree in which all the registered certificates are stored in lexicographic order. This Root is generated by the ILS from the tree and sent to the CA2 but the intruder can get this value in between and can send it to CA2 but the intruder cannot change the Root value so it is not an active attack.

- Analysis using ProVerif

Analysis of the three protocols of ARPKI gave following results. In the specification of registration and updation process, there are only authentication queries as there is no secret data. Analysis of registration process has given following result.

In both the results, it is found that authentication of CA1 to ILS1 and authentication of A to CA1 are violated. Violation of authentication of A to CA1 was also detected by AVISPA. Other authentication queries are either true or they cannot be proved. Analysis of confirmation process has given following result.

The secrecy of the message root is true, but the authentication of CA2 to CA1 and authentication of ILS1 to CA2 are violated. The violation of the same authentication of ILS1 to CA2 is detected by AVISPA.

Authentication means, suppose B authenticates A on nonce Nb then it means when B receives the message, he can be sure that A has sent Nb. In authentication loss the message cannot be changed in between. One may try to replay such messages to create problems.

C. Analysis of OPT

- Analysis using AVISPA

The OPT protocol is safe. Analysis shows that it does not contain vulnerabilities for an attack.

In both the DRKey protocols (i.e. with trust and without trust) the intruder can form a man-in-the-middle attack.

Intruder can get the data which is sent by S to router R1. It is a passive attack but it violates the authentication property. In the attack trace, it is found that intruder can get the data sent by the S (Source). This data is supposed to reach to R1 (Router1) but in between intruder can get this data and can forward it to R1.

- Analysis using ProVerif

When all the three protocols i.e. DRKey with trust, DRKey without trust and OPT are analyzed using ProVerif, it is found that all the specified goals in each protocol are reachable. There is neither any false authentication correspondence nor any violated secrecy query found in any of these protocols. So it is possible to say that the specifications of protocols are safe and don't contain any possibilities of attacks.

D. Comparison

Table 1 shows the results got after analysis of protocols and the results previously obtained by other tools for the same protocols.

Table 1. Comparison of Results

Name	AKI	ARPKI	OPT
AVISPA	Intruder can get the certificate for any domain	In Registration and Updation process intruder can get the certificate. In confirmation process authentication of Root value fails	In DRKey protocols Intruder can form man-in-the-middle attack.(Passive attack) OPT is safe
ProVerif	Authentication of CA to A violated, this is same as the result of AVISPA	Results are same as that of AVISPA	All the protocols are safe
TAMARIN Prover	Not Verified	Compromising 3 or more entities (i.e. CA1, CA2 and ILS1) will give malicious certificate	Not Verified
Coq Theorem Porver	Not Verified	Not Verified	Two malicious routers can trick other router by changing the source

E. Characteristics of AVISPA and ProVerif

The characteristics of AVISPA and ProVerif are as follows.

Characteristics of AVISPA

- **Input**
  - The protocol is specified using the HLPSL language
  - The communicating parties are modeled as roles
  - The anticipated intruders need not to be specified as agents
- **User interface**
  - Tool has its a graphical user interface
- **Session**
  - It is possible to run the protocol for both bounded and unbounded number of sessions case
- **Output**
  - It generates the following possible outputs. Protocol holds for n fixed Depth, Protocol is false and attack trace is shown, Protocol holds for all traces
  - Attack traces are generated which give a visual flow of a trace. Traces are self explanatory
- **Other**
  - Tool by its own discretion cannot check for secrecy of all possible variables, without explicit claims, they are necessary
  - All possible trace patterns are generated depicting protocol execution

Characteristics of ProVerif

- **Input**
  - The protocol is specified using Horn clauses or pi calculus
  - The communicating parties are modeled as processes
  - In ProVerif also intruders need not to be specified as agents
- **User interface**
  - Tool has no graphical user interface, it has to be run using command line interface
- **Session**
  - It is possible to run the protocol only for unbounded number of session case
- **Output**
  - It generates the following outputs. Protocol is true, Protocol is false and the attack trace is generated, Protocol cannot be proven when false attack is found, and also Tool might not terminate for some cases
  - Step by step trace is generated explaining the run and attack
- **Other**
  - It checks only those attacks for which the query has been specified in the code
  - Here also all possible trace patterns are generated when the protocol terminates
  - To check equality if...then or let...in can be used

V. Conclusion

Cryptographic protocols specification and verification is important because of the powerful intruders present nowadays. There are various tools present for the purpose of analysis of cryptographic protocols such as ProVerif, Scyther, AVISPA, Athena etc. Specification and verification of cryptographic protocols should be done by possible tools to avoid flaws in the protocols.

In this experimental work, specification of the protocols namely AKI, ARPKI and OPT is done as per our interpretation of protocols and their analysis has given following results.

1. When AKI is analyzed using AVISPA; it is found that Intruder can get the certificate for any domain. ProVerif's result is same as that of AVISPA.
2. Analysis of ARPKI using both tools gave same result that is, in Registration and Updation process intruder can get the certificate. In confirmation process authentication of Root value fails.
3. Analysis of OPT protocol using AVISPA gave the result that, in both DRKey protocols Intruder can form man-in-the-middle attack, this is passive attack and OPT protocol is safe. OPT When analyzed with ProVerif gave the result that all the protocols are safe.

Future Work:

The proposed work can be extended by verifying these experimented protocols using other tools like Scyther, Isabelle, TAMARIN, Coq theorem prover, Athena etc to have wider evaluation.

References

[1] C. Meadows, “Formal methods for cryptographic protocol analysis: emerging issues and trends,” *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 44–54, 2003. doi: 10.1109/JSAC.2002.806125.

[2] B. Blanchet, “Automatic verification of correspondences for security protocols,” *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, 2009.

[3] L. Viganò “Automated security protocol analysis with the AVISPA tool,” *Electr. Notes Theor. Comput. Sci.*, vol. 155, pp. 61–86, 2006. doi: 10.1016/j.entcs.2005.11.052.

[4] C. J. F. Cremers, “The scyther tool: Verification, falsification, and analysis of security protocols,” in *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, 2008, pp. 414–418. doi: 10.1007/978-3-540-70545-1_38.

[5] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, “The TAMARIN prover for the symbolic analysis of security protocols,” in *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, 2013, pp. 696–701. doi: 10.1007/978-3-642-39799-8_48.

[6] D. X. Song, S. Berezin, and A. Perrig, “Athena: A novel approach to efficient automatic security protocol analysis,” *Journal of Computer Security*, vol. 9, no. 1/2, pp. 47–74, 2001.

[7] C. Meadows, “The NRL protocol analyzer: An overview,” *J. Log. Program.*, vol. 26, no. 2, pp. 113–131, 1996.

[8] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil D. Gligor. Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17*, 2013, pages 679-690, 2013.

[9] David A. Basin, Cas J. F. Cremers, Ti_any Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7*, 2014, pages 382-393, 2014.

[10] Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. Lightweight source authentication and path validation. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22*, 2014, pages 271-282, 2014.

[11] J. K. Millen, S. C. Clark, and S. B. Freedman, “The interrogator: Protocol security analysis,” *IEEE Trans. Software Eng.*, vol. 13, no. 2, pp. 274–288, 1987.

[12] L. C. Paulson, “Isabelle: The next 700 theorem provers,” in *Logic and computer science*, vol. 31, 1990, pp. 361–386.

[13] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, “The coq proof assistant reference manual: Version 6.1,” 1997.

[14] E. M. Clarke, S. Jha, and W. R. Marrero, “Verifying security protocols with brutus,” *ACM Trans. Softw. Eng. Methodol.*, vol. 9, no. 4, pp. 443– 487, 2000.

[15] J. C. Mitchell, M. Mitchell, and U. Stern, “Automated analysis of cryptographic protocols using mur-phi,” in *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, 1997, pp. 141–151.

Authors’ Profiles

Amol H. Shinde is a student at Walchand College of Engineering, Sangli, Maharashtra. He has completed his B.E. in Computer Science and Engineering from Shivaji University, Kolhapur (Maharashtra) in 2012 and currently pursuing his M.Tech in Computer Science and Engineering specialization in Information Technology at Walchand College of Engineering. His research areas are security protocol verification

A. J. Umbarkar is presently working as an Assistant Professor in Information Technology department at Walchand College of Engineering, at Sangli, MS, India. He has 13 years of teaching experience. His research interests include Function Optimization, Parallel Evolutionary Algorithms and Parallel programming. He has published 20 research papers in Conferences and Journals.