NIST Technical Note 2123

Formal Verification of Bootstrapping Remote Secure Key Infrastructures (BRSKI) Protocol Using AVISPA

Monika Singh
Mudumbai Ranganathan

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2123

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Technical Note 2123

Formal Verification of Bootstrapping Remote Secure Key Infrastructures (BRSKI) Protocol Using AVISPA

Monika Singh
Mudumbai Ranganathan
Advanced Network Technologies Division
Information Technology Laboratory
National Institute of Standards and Technology

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2123

October 2020

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology Technical Note 2123
Natl. Inst. Stand. Technol. Tech. Note 2123, 24 pages (October 2020)
CODEN: NTNOEF

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2123

Abstract

The last decade has seen significant growth in the number of IoT devices. These devices can onboard the network and connect to each other. The process through which a new IoT device connects to the network and subsequently enables its services is called *bootstrapping*. A single entity connecting large numbers of new IoT devices to networks makes manual bootstrapping infeasible. It requires an automated system to enable a new device to be located and securely onboard the network. The Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol is one of the well-known protocols that provides a way for secure device onboarding. In this work, we present the first formal security analysis of the BRSKI protocol using a verification tool called AVISPA (Automated Validation of Internet Security Protocols and Applications). AVISPA provides a formal security validation of any network protocol by building and analyzing the formal security models of that protocol's operations.

Key words

Authentication, AVISPA, Bootstrapping, HLPSL, SPAN, Verification, X.509 certificates.

Table of Contents

List of Tables

List of Figures

1. Introduction

The continued rapid growth in the number of IoT devices requires an automatic and secure technique that enables the devices to securely bootstrap trust between other devices and the networks on which they connect and communicate. Several newly emerging protocols are addressing this issue. Bootstrapping Remote Secure Key Infrastructures (BRSKI) [1] is one of such protocol that provides a solution to bootstrap IoT devices in the context of the Autonomic Networking Integrated Model and Approach (ANIMA) [2]. BRSKI facilitates the authentication of the device to the network operator and vice-versa with the help of the device manufacturer using the IEEE standard 802.1AR secure device identity [3]. The protocol is built on top of HTTP and transport layer security (TLS). The new device requires only link-local connectivity and does not have a network routable address until authenticated.

The BRSKI protocol uses cryptographic primitives to ensure the security of the protocol. The major issue is that it is very difficult to analyze the security properties of a protocol no matter its size. A classic example could be the Needham-Schroeder Public-Key Protocol which had been proven secure by Burrows et.al. [4] until several years later when Gavin Lowe [5][6] showed a triangular attack on the protocol using an automated tool FDR/Casper [7] [8]. This example explains the extent of the complexity of protocol analysis and the importance of automatic analysis to prove the security of the protocol and gain wider acceptance. Over the decades various automated protocol analysis tools based on formal analysis has been presented such as AVISPA [9], proverif [10], casper/FDR [8], Scyther [11], tamarin [12], etc.

In this work, we use the *Automated Validation of Internet Security Protocols and Applications (AVISPA)* tool to formally verify the security property of BRSKI. AVISPA is a push-button verification tool developed as a collaboration between the Artificial Intelligence Laboratory at DIST (University of Genova, Genova, Italy), the CASSIS group at INRIA Lorraine (LORIA, Nancy, France), the Information Security group at ETHZ (Zürich, Switzerland), and Siemens AG (Munich, Germany). AVISPA comprises four distinct formal verification approaches that can formally validate the security property of a protocol (i.e. On-the-fly Model-Checker, Constraint-Logic-based Attack Searcher, SAT-based Model-Checker and Tree Automata-based Protocol Analyser). It uses the High-LevelProtocol Specification Language (HLPSL) to specify the protocol and its security properties in order to use all four analysis techniques.

The rest of the paper is structured as follows: the notation, definition, and BRSKI protocol design are described in section 2. The section 3 presents the modeling, formal verification, and validation results. We conclude the paper in section 4.

2. Bootstrapping Remote Secure Key Infrastructures (BRSKI)

2.1 Notation and Definitions

- Sn : denotes the pledge's iDevID serial-number

- Assr : represents the assertion
- Crdate : Date and time the voucher request was created on
- N_p : Nonce generated by the pledge
- proximity-registrar-cert : contains the registrar's TLS certificate shared during the TLS session
- Issuer : denotes the issuer of the pledge's iDevID certificate
- prior-signed-voucher-request : contains the pledges voucher request
- pinned-domain-cert : represents the join domain's CA certificate
- sln indicates the subjectAltName of iDevID

2.2 Protocol Design

The Bootstrapping Remote Secure Key Infrastructures (BRSKI) protocol provides a solution for a resource-constrained new device to automatically onboard the correct network in a secure manner. BRSKI specification refers to the new device as the *pledge*. The protocol aims to establish a trusted relationship between the pledge and the network operator/owner referred as the *registrar* in such a way so that the registrar and pledge can assure and authenticate each other's identity. This is done using a 802.1AR iDevID [3] cert which installed into the device by the manufacturer during the manufacturing process. The iDevID cert indicates the manufacturer, type and serial number of the given device. The manufacturer also installs the trust anchor for the manufacturer's authorized signing authority (known as the *MASA*) at compile time which the device uses to authenticate the *MASA*.

To initiate the bootstrapping process, the pledge sends the iDevID as part of TLS session and establish a provisional TLS connection through a join proxy. Once the provisional TLS connection is established the pledge sends a signed voucher-request [13] to the registrar which includes information about the pledge such as the assertion (Assr), nonce (N_p),[1] serial-number (Sn), created-on (Crdate), and the proximity-registrar-cert (which is the registrar's TLS certificate shared during the TLS session). While receiving the pledge voucher-request, the registrar determines if it is expecting such device and if yes then the registrar locates the device's MASA and sends that MASA a signed registrar voucher-request that contains the entire pledge voucher-request. The registrar's voucher-request includes the following information: Assr, N_p, Sn, Crdate from the pledge voucher-request, and idevid-issuer (Issuer) from the iDevID certificate and the entire pledge voucher-request in prior-signed-voucher-request field. Then MASA checks its internal device database with respect to the provided device serial number in the voucher-request. If the voucher-request

[1] In this analysis, we have analyzed voucher-request with the nonce.

is accepted, a voucher is issued. The voucher contains assertion (Assr), nonce (N_p), serial-number (Sn) and pinned-domain-cert (domain's CA certificate). The registrar redeems the voucher by passing it along to the pledge. The pledge validates the signed voucher using the pre-installed *MASA*'s trust anchor. The pledge also verifies the registrar using the pinned-domain cert and completes the authentication of the provisional TLS connection. The pledge returns the voucher telemetry status indicating the voucher acceptance status. A successful voucher validation indicates an established, trusted relationship between the pledge and the registrar. Figure 1 represents the overview of the BRSKI protocol. The detailed protocol simulation is shown in figure 3.

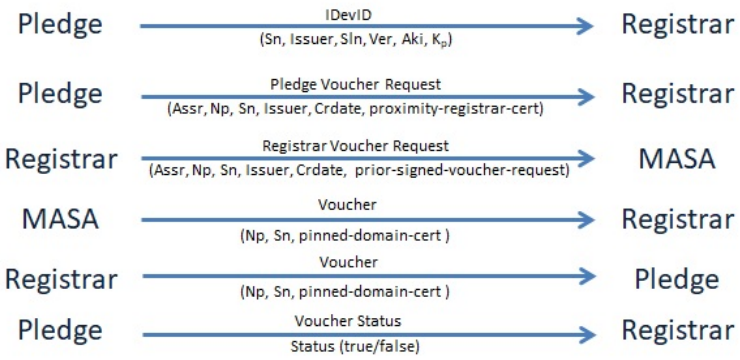

Fig. 1. BRSKI overview

3. BRSKI Validation with AVISPA and SPAN

The *Automated Validation of Internet Security Protocols and Applications (AVISPA)* is a formal protocol verification tool that can be used to build, analyze, and validate security properties of network security protocols. AVISPA integrates four different verification back-end tools to implement a variety of approaches to analyze the protocols.

- OFMC (On-the-fly Model-Checker) uses number of symbolic techniques to represent the state-space to perform protocol falsification and verification for the boundless number of sessions in a demand-driven fashion.
- CL-AtSe (Constraint-Logic-based Attack Searcher) is a constraint based approach. It uses some simplification and redundancy elimination techniques to integrate a new specification for cryptographic functions.
- SATMC (SAT-based Model-Checker) use SAT solver to generate security property violation and attacks.
- TA4SP (Tree Automata-based Protocol Analyser) validates the security properties by estimating the intruder's knowledge using regular tree languages for the unbounded number of sessions.

Each back-end tool uses different techniques to perform verification for a finite and infinite number of sessions. In order to analyze any protocol in AVISPA and SPAN, it must be modeled in a modular and formal language called *High-Level Protocol Specification Language (HLPSL)*. SPAN is another tool that has been used to test the correctness of the protocol simulation explained in section 3.2.

3.1 BRSKI specification in HLPSL

BRSKI HLPSL specification is presented in appendix 4. The specification has three agents: the pledge (P), the registrar (R) and the Manufacturer Authorized Signing Authority also known as MASA (M). HLPSL is a role-based language so we have specified roles of each agent in the protocol using a set of variables, constant, and transitions (called basic roles). Where transitions define the message exchanges between two agents. Then two composite roles are defined known as session and environment. The Composite roles instantiate basic roles to model the entire protocol. A session integrates all roles together to run a valid session of a protocol. Each run of the protocol is a session. For example, a session of BRSKI in our specification is defined as follows:

$$session(p, r, m, kp, kr, km, kca, keygen, prf, sn,$$
$$issuer, aki, sln)$$

The session is parameterized by all variables necessary for one session, which is all agents (p-pledge, r-registrar, m-MASA) public keys (kp, kr, km), functions (keygen, prf-used during TLS session to generate session keys), and iDevIDcert values (sn-serial number, issuer-idevid issuer, aki-authority key identifier, sln-subjectAlternateName). Each session runs in the environment role including initial intruder knowledge. Once all roles are defined the list of security properties, which are to be verified, are declared in the goal section. Using AVISPA we can only test for Confidentiality, Authentication, Freshness (Anti-replay). Hence, in this paper, we test BRSKI for these three properties.

3.2 HLPSL correctness verification

The semantic and syntactic correctness of HLPSL modeling can be validated using the SPAN (Security Protocol ANimator) tool. SPAN provides a graphical interface that helps to debug the HLPSL specification. The syntactic and semantic error of HLPSL specification can be identified and rectified in the following two ways:

3.2.1 Debugging of Syntax error

In order to execute the HLPSL specification, SPAN translates it into the *Intermediate Format (IF)* using tool HLPSL2IF as shown in figure 2. This process results in either the

syntax errors in the HLPSL file or a '.IF' file which is later used in the execution of the specification. Our implementation has been successfully verified using tool HLPSL2IF.

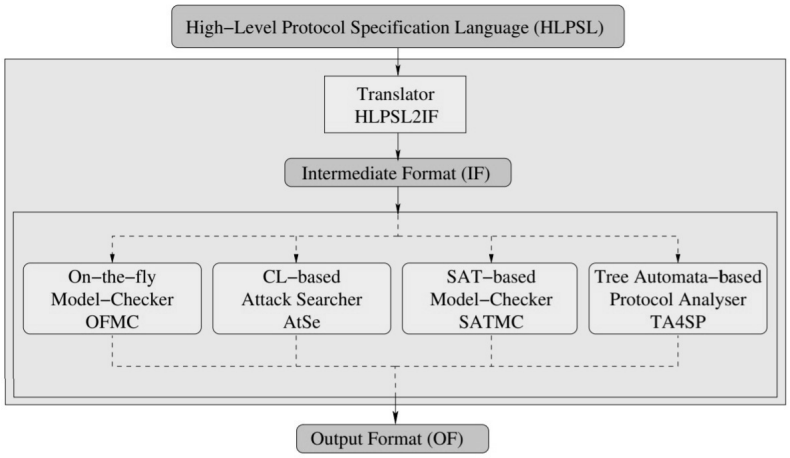

Fig. 2. Tool architecture

3.2.2 Debugging of Semantic

The semantic of the protocol is verified using the 'Protocol simulation' option of the SPAN, which allows to visualize the step by step message exchange of the protocol. This option is very useful to fix the semantic errors in the HLPSL specification. Figure 3 shows the BRSKI protocol simulation obtained from SPAN. The visualization of the complete protocol message exchange implies no semantic error in the HLPSL specification.
In figure 3 steps 1 to 4 represents the TLS handshake and generation of the TLS shared secret key, which will be used for further communication between the pledge and the registrar. Step 5 shows the pledge voucher-request. Step 6 to 9 represents the TLS handshake between the registrar and MASA and step 10 shows the registrar voucher-request to MASA. Step 10 and 11 present the voucher exchange between the MASA to the registrar and the registrar to the pledge. After successful debug, we tested our BRSKI HLPSL implementation on all four verification back-end tools. We present the results obtained by each tool in the next section.

3.3 BRSKI security validation

Using AVISPA and SPAN we analyze the following security properties of BRSKI.

- **Authentication** Authentication is the process by which both participants in a communication session ensure that they are communicating with the desired party. Three parties are involved in this protocol Pledge, Registrar and MASA. Mutual authentication is established in the protocol in the following way. The pledge is authenticated

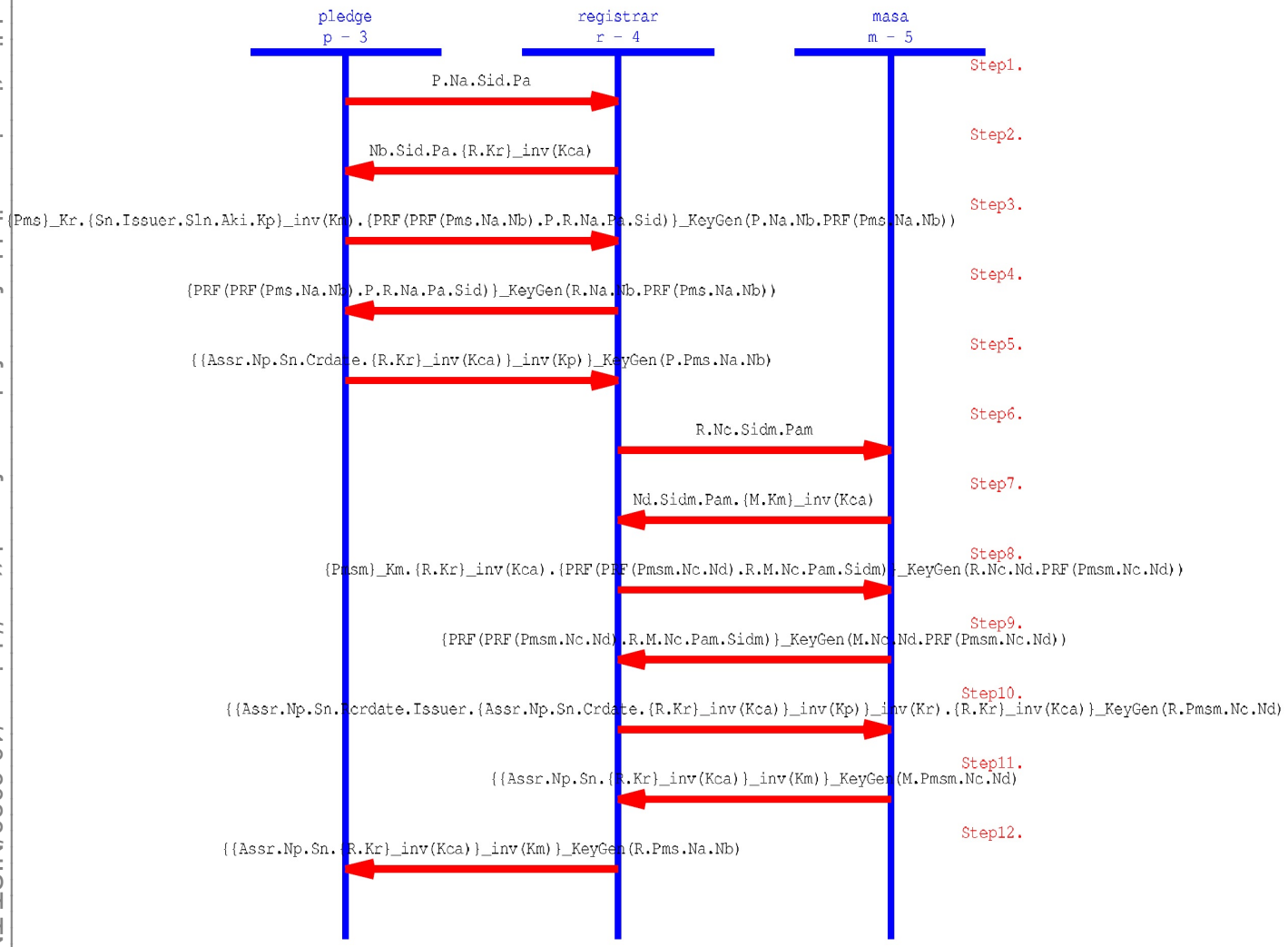

Fig. 3. BRSKI protocol simulation on SPAN

via its iDevID certificate and the pledge provisionally authenticates the registrar during the TLS handshake with the registrar's unauthenticated server certificate. Later, after receiving the voucher, the pledge completes the registrar's certificate authentication using the pinned-domain-cert field of the voucher. The pledge validates each message received until a voucher is verified. The registrar and the MASA authenticate each other during a normal TLS handshake. The MASA authenticates the registrar's voucher-request by verifying the registrar's signature over the voucher-request message. The MASA authenticates the pledge using the contents of the attached iDevID certificate (e.g. the serial number, etc.) in the voucher-request. The pledge verifies the voucher signature using the MASA's pre-installed trust anchor and verifies the Sn and Np.

In the HLPSL specification, the authentication property of a protocol can be tested using *request* and *witness* clause, which allows the participating agents to assert that they want to be the peer of another agent and will agree on a value (variable) for authentication.

Here we show an example of the request-witness pair of the pledge authenticating a voucher using the nounce and the serial number, which is later declared as the authentication goal in the goal section of HLPSL.

$$M:$$
$$witness(M,P,auth\_np,Np')$$
$$witness(M,P,auth\_sn,Sn)$$

Which indicates that the MASA hopes the Pledge can authenticate it on Np and Sn. 'auth_np' and 'auth_sn' represent the protocol_id later used to set the goal.

$$P:$$
$$request(P,M,auth\_np,Np)$$
$$request(P,M,auth\_sn,Sn)$$

Which refers that the Pledge requests the MASA to authenticate itself on Np and Sn.

$$authentication\_onauth\_np$$
$$authentication\_onauth\_sn$$

Similarly, we have validated the authentication of each entity and message in our specification shown in appendix 4.

- **Confidentiality / Secrecy**

  This property implies that the secret information is not revealed to an unauthorized party during the message exchange in the protocol. BRSKI retains confidenciality by having the message exchange over a TSL-encrypted channel. The pledge and the registrar perform their message exchange over a provisional TLS channel and the registrar and the MASA communicate over a secure TSL-channel. In AVISPA the confidentiality of the secret values can be validated by declaring the secrecy clause in the role where secret information is generated and adding the secrecy goal to the goal of HLPSL specification. For example, we tested the confidentiality of Np by adding the following clause to the Pledge role.

  $$secret(Np, sec\_np\_pr, P, R)$$

  Where the first argument represents the secret value, the second argument specifies the secrecy goalID and the last parameter represents the agents between whom the secret value is shared. Here we test the confidentiality of Np between the pledge and the registrar by setting following goal

  $$secrecy\_of\ sec\_np\_pr$$

  Similarly, we have tested confidentiality of shared keys and other secret values in BRSKI.

- **Replay Attack** A replay attack occurs when an unauthorized agent or intruder captures the network traffic and maliciously delays or repeats it while impersonating the legitimate agent. Replay attacks can be prevented by adding some element of freshness to the messages which can be the sessionID, nonce or timestamp. BRSKI prevents this attack by including a nonce in the voucher-request and the sessionId & nonce during TLS handshake.

  In order to test against a replay attack, we have considered the following four protocol sessions.

  1. session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln): This session runs the protocol considering all participating agents in the session are legitimate agents.
  2. session(i,r,m,ki,kr,km,kca,keygen,prf,sn,issuer,aki,sln): This session runs a scenario where an intruder plays the role of the agent *pledge*.
  3. session(p,i,m,kp,ki,km,kca,keygen,prf,sn,issuer,aki,sln): This session represents the protocol run for the scenario where an intruder impersonates the legitimate agent registrar.
  4. session(p,r,i,kp,kr,ki,kca,keygen,prf,sn,issuer,aki,sln): This session runs the protocol with an intruder impersonating the legitimate agent MASA.

The AVISPA tool only supports the Dolev-Yao model. In the Dolev-Yao intruder model, the intruder has full control over the network, i.e. all messages sent by agents go to the intruder. The intruder may intercept, analyze, and/or modify the message and send any message to an agent pretending to be any other agent but the impostor cannot encrypt or decrypt without the knowledge of the key.

- **Verification results:** The results obtained from validating BRSKI with OFMC and CL-AtSe backends are shown in the next Tables 2. In the case of SATMC, the result was always "Inconclusive". The protocol can not be tested with TA4SP because of some 'technical issues about non-left-linearity in term rewriting with tree automation'. Hence, SATMC and TA4SP were not used in any further analysis. Table 1 below shows the scenarios that have been analyzed to find possible security goal violations.

Table 1. Attack scenarios

Scenario	Session Configuration
cfg1	session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln)
cfg2	session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln) session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln)
cfg3	session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln) session(i,r,m,ki,kr,km,kca,keygen,prf,sn,issuer,aki,sln)
cfg4	session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln) session(p,i,m,kp,ki,km,kca,keygen,prf,sn,issuer,aki,sln)
cfg5	session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln) session(p,r,i,kp,kr,ki,kca,keygen,prf,sn,issuer,aki,sln)
cfg6	session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln) session(i,r,m,ki,kr,km,kca,keygen,prf,sn,issuer,aki,sln) session(p,i,m,kp,ki,km,kca,keygen,prf,sn,issuer,aki,sln) session(p,r,i,kp,kr,ki,kca,keygen,prf,sn,issuer,aki,sln)

Configuration cfg1 aims to find attack in a normal session where all agents are legitimate. The rest of the configuration aims to identify a reply attack. In order to test the replay attack, we run two or more coherent parallel sessions of the protocol. Configuration cnf2 runs two sessions with all legitimate/legit participants. Configuration cnf3, cnf4, cnf5 runs two coherent parallel sessions where one session is a normal session with all legitimate agents and the other session with an intruder impersonating one of the legitimate participants. The aim is to find possible attacks in a session when carried out from an intruder running in another different session. Configuration cfg5 runs four parallel sessions where one session is with all legit participants and the rest have an intruder impersonating as a legitimate agent pledge, registrar, and

MASA respectively. Note that based on the role imitated by the intruder its knowledge varies in each configuration.

Table 2 presents the obtained results. AVISPA summarizes the result using one of the following:

- **Safe** indicates that the protocol does not violate any security goal specified in the HLPSL specification.
- **Unsafe** indicates that there is a security flaw in the protocol for which an attack trace has been found.
- **Inconclusive** refers that due to some underlying issues (i.e. TIME_OUT, MEMORY_OUT, NOT_SUPPORTED, etc.) AVISPA can not analyze the protocol.

As table 2 shows that for cfg1 AVISPA results that protocol is safe using both the OFMC and CL-AtSe backends. It indicates that there are no authentication and secrecy attack on BRSKI. Similarly, cfg2-cfg6 presents that protocol is protected against a replay in multiple parallel sessions with impersonating intruder. We tested cfg2, cfg3,cfg4, cfg5, and cfg6 with the OFMC & CL-AtSe backend and found that the BRSKI protocol is safe in all scenarios.

4. Conclusion and Future Work

This work examines the security properties of one of the prominent and well know device bootstrapping protocols, *Bootstrapping Remote Secure Key Infrastructures (BRSKI)* using automated security verification tools *AVISPA (Automated Validation of Internet Security Protocols and Applications)* and *SPAN (Security Protocol ANimator for AVISPA)*. From the formal verification, we conclude that the protocol satisfies the specified security properties (secrecy, authentication, and freshness) and it is secure against active and passive attacks. This Paper also presents the HLPSL specification of BRSKI that can be used to further simulate, analyze, and validate the protocol using SPAN and AVISPA. In future we would like to validate another prominent device onboarding protocol such as DPP (Device Provisioning Protocol) [14] using AVISPA. We also intend to explore the verification of device bootstrapping protocols using other existing verification tools such as proverif, Scyther, etc.

References

[1] Pritikin M, Richardson M, Behringer M, Bjarnason S, Watsen K (2019) Bootstrapping remote secure key infrastructures (brski). *Internet-Draft draft-ietf-anima-bootstrapping-keyinfra-18, IETF* .

Table 2. Validation results with OFMC and CL-AtSe

Analysis scenario	Tool	Description	Result
cfg1	OFMC	VisitedNodes:9 nodes Depth: 4 plies Search Time: 0.03s	Safe
	CL-AtSe	Analysed : 18 states Reachable : 5 states Translation: 0.07 seconds Computation: 0.00 seconds	Safe
cfg2	OFMC	Visited Nodes: 1016 nodes depth: 8 plies Search Time: 3.76s	Safe
	CL-AtSe	Analysed : 3918 states Reachable : 652 states Translation: 0.21 seconds Computation: 0.11 seconds	Safe
cfg3	OFMC	visitedNodes: 150 nodes depth: 6 plies searchTime: 0.44s	Safe
	CL-AtSe	Analysed : 655 states Reachable : 130 states Translation: 0.16 seconds Computation: 0.02 seconds	Safe
cfg4	OFMC	visitedNodes: 233 nodes depth: 7 plies searchTime: 0.70s	Safe
	CL-AtSe	Analysed : 560 states Reachable : 111 states Translation: 0.13 seconds Computation: 0.01 seconds	Safe
cfg5	OFMC	visitedNodes: 272 nodes depth: 7 plies searchTime: 0.92s	Safe
	CL-AtSe	Analysed : 655 states Reachable : 130 states Translation: 0.18 seconds Computation: 0.02 seconds	Safe
cfg6	OFMC	visitedNodes: 0 nodes depth: 1000000 plies searchTime: 9.95s	Safe
	CL-AtSe	Analysed : 3804750 states Reachable : 422749 states Translation: 0.42 seconds Computation: 115.38 seconds	Safe

[2] Infrastructure SAN (2018) Autonomic networking integrated model and approach (anima). *Emerging Automation Techniques for the Future Internet* :90.
[3] (2009) Ieee 802.1ar secure device identifier. Available at https://standards.ieee.org/standard/802_1AR-2009.html.
[4] Burrows M, Abadi M, Needham RM (1989) A logic of authentication. *Proceedings of the Royal Society of London A Mathematical and Physical Sciences* 426(1871):233–271.
[5] Lowe G (1996) Breaking and fixing the needham-schroeder public-key protocol using fdr. *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (Springer), , pp 147–166.
[6] Lowe G (1995) An attack on the needham- schroeder public- key authentication protocol. *Information processing letters* 56(3).
[7] Roscoe B (1994) *A classical mind: essays in honour of CAR Hoare*, .
[8] Lowe G (1998) Casper: A compiler for the analysis of security protocols. *Journal of computer security* 6(1-2):53–84.
[9] Armando A, Basin D, Boichut Y, Chevalier Y, Compagna L, Cuéllar J, Drielsma PH, Héam PC, Kouchnarenko O, Mantovani J, et al. (2005) The avispa tool for the automated validation of internet security protocols and applications. *International conference on computer aided verification* (Springer), , pp 281–285.
[10] Blanchet B (2009) Automatic verification of correspondences for security protocols. *Journal of Computer Security* 17(4):363–434.
[11] Cremers CJ (2008) The scyther tool: Verification, falsification, and analysis of security protocols. *International conference on computer aided verification* (Springer), , pp 414–418.
[12] Meier S, Schmidt B, Cremers C, Basin D (2013) The tamarin prover for the symbolic analysis of security protocols. *International Conference on Computer Aided Verification* (Springer), , pp 696–701.
[13] Watsen K, Richardson M, Pritikin M T. eckert," a voucher artifact for bootstrapping protocols (RFC 8366, DOI 10.17487/RFC8366, May 2018,¡ https://www. rfc-editor. org/info . . .),
[14] Alliance WF (2018) Device provisioning protocol specification version 1.0. *Tech. Rep.* (Wi-Fi Alliance), , .

Appendix A: HLPSL specification of BRSKI[2]

```
%% HLPSL:

role pledge (P, R, M : agent,
Kr, Km, Kca          : public_key,
KeyGen, PRF          : hash_func,
```

[2]https://github.com/usnistgov/BRSKI-HLPSL

```
Sn                    : text, %devID serial Number
Issuer                : text, %devID issuer
Aki                   : text, %devID authorityKeyIdentifier
Sln                   : text, %devID subjectAltName
SND_R, RCV_R, SND_M, RCV_M   : channel (dy))
played_by P def=

local Np, Na, Nb, Assr, Pms, Sid, Pa : text,
Crdate                          : text,
State                           : nat,
Kp                              : public_key,
Finishedp     : hash(hash(text.text.text).agent.agent.text.text.text),
ClientKp, ServerKp  : hash(agent.text.text.hash(text.text.text)),
Mp     : hash(text.text.text)

const sec_np_pr, auth_np, auth_sn, auth_na_nb, auth_reg_cert, auth_na_nb2,
auth_idev, auth_reg_certchain, tls_pledgek, tls_registrark : protocol_id%,

init State := 0

transition

0. State  = 0 /\ RCV_R(start) =|>
State':= 2
/\ Na' := new()
/\ Pa' := new()
/\ Sid' := new()
/\ SND_R(P.Na'.Sid'.Pa')


2. State  = 2 /\ RCV_R(Nb'.Sid.Pa.{R.Kr'}_inv(Kca)) =|>
State':= 4 /\ Pms' := new()
/\ Mp' := PRF(Pms'.Na.Nb')
/\ Finishedp' := PRF(Mp'.P.R.Na.Pa.Sid)
/\ ClientKp' := KeyGen(P.Na.Nb'.Mp')
/\ ServerKp' := KeyGen(R.Na.Nb'.Mp')
/\ SND_R({Pms'}_(Kr).{Sn.Issuer.Sln.Aki.Kp}_inv(Km).{Finishedp'}_ClientKp')
/\ secret(ClientKp,tls_pledgek,{P,R})
/\ secret(ServerKp,tls_registrark,{P,R})

/\ witness(P,R,auth_na_nb,Na.Nb')
/\ witness(P,M,auth_idev,Sn) %Masa can authenticate pledge on serialNumber
```

```
4. State  = 4 /\ RCV_R( {PRF(PRF(Pms.Na.Nb).P.R.Na.Pa.Sid)}
   _KeyGen(R.Na.Nb'.PRF(Pms.Na.Nb))) =|>
State':= 6 /\ Np'   := new()
/\ Assr' := new()
/\ Crdate' := new()
/\ SND_R({{Assr'.Np'.Sn.Crdate'.{R.Kr}_inv(Kca)}_(inv(Kp))}
         _(KeyGen(P.Pms.Na.Nb')) )
/\ secret(Np,sec_np_pr,{P,R})
/\ request(P,R,auth_na_nb2,Na.Nb)
/\ witness(P,R,auth_reg_cert,{R.Kr}_inv(Kca))

6. State  = 6 /\ RCV_R({{Assr'.Np.Sn.{R.Kr'}_inv(Kca)}_(inv(Km))}
   _(KeyGen(R.Pms'.Na.Nb'))) =|>
State':= 8 /\ request(P,R,auth_reg_certchain,{R.Kr'}_inv(Kca))
/\ request(P,M,auth_np,Np)
/\ request(P,M,auth_sn,Sn)

end role

role registrar (P, R, M    : agent,
Kr, Km, Kca                : public_key,
KeyGen, PRF                : hash_func,
SND_P, RCV_P, SND_M, RCV_M : channel (dy))
played_by R def=

local Assr, Np, Na, Nb, Nc, Nd,  Pms, Pmsm, Sid, Pa, Sidm, Pam  : text,
Sn, Crdate, Rcrdate     : text,
State                   : nat,
Kp                      : public_key,
Issuer                  : text,
Aki                     : text,  %authorityKeyIdentifier
Sln                     : text,  %subjectAltName
Finishedp, Finishedm : hash(hash(text.text.text).agent.agent.
                       text.text.text),
ClientKp, ServerKp, ClientKm, ServerKm: hash(agent.text.text.
                                        hash(text.text.text)),
Mm : hash(text.text.text)


const sec_np_rm, auth_na_nb, auth_na_nb2, auth_reg_cert, auth_np, auth_np1,
```

```
auth_nc_nd, auth_reg_certchain, auth_nc_nd2, tls_registrarrmk, tls_masarmk:
protocol_id%,

init State := 1

transition

1. State  = 1 /\ RCV_P(P.Na'.Sid'.Pa') =|>
State':= 3 /\ Nb' := new()
/\ SND_P(Nb'.Sid'.Pa'.{R.Kr}_inv(Kca))
/\ witness(R,P,auth_na_nb2,Na'.Nb')

3. State  = 3
/\ RCV_P({Pms'}_Kr.{Sn'.Issuer'.Sln'.Aki'.Kp'}_(inv(Km)).
         {Finishedp'}_ClientKp')
/\ Finishedp = PRF(PRF(Pms'.Na.Nb').P.R.Na.Pa.Sid)
/\ ClientKp = KeyGen(P.Na.Nb'.PRF(Pms'.Na.Nb'))=|>

State':= 9
/\ ServerKp' := KeyGen(R.Na.Nb'.PRF(Pms'.Na.Nb'))
/\ SND_P( {PRF(PRF(Pms'.Na.Nb').P.R.Na.Pa.Sid)}
          _KeyGen(R.Na.Nb'.PRF(Pms'.Na.Nb')) )
/\ request(R,P,auth_na_nb,Na.Nb)

9. State  = 9
/\ RCV_P({{Assr'.Np'.Sn.Crdate'.{R.Kr'}_inv(Kca)}_(inv(Kp))}
   _(KeyGen(P.Pms'.Na.Nb'))) =|>
State':= 11
/\ Nc' := new()
/\ Pam' := new()
/\ Sidm' := new()
/\ SND_M(R.Nc'.Sidm'.Pam')

11. State  = 11
/\ RCV_M(Nd'.Sidm'.Pam'.{M.Km}_inv(Kca)) =|>
State':= 13
/\ Pmsm' := new()
/\ Mm' := PRF(Pmsm'.Nc.Nd')
/\ Finishedm' := PRF(Mm'.R.M.Nc.Pam.Sidm)
/\ ClientKm' := KeyGen(R.Nc.Nd'.Mm')
/\ ServerKm' := KeyGen(M.Nc.Nd'.Mm')
```

```
/\ SND_M({Pmsm'}_(Km).{R.Kr}_inv(Kca).{Finishedm'}_ClientKm')
/\ witness(R,M,auth_nc_nd2,Nc.Nd')
/\ secret(ClientKm,tls_registrarrmk,{R,M})
/\ secret(ServerKm,tls_masarmk,{R,M})


13. State  = 13
/\ RCV_M({PRF(PRF(Pmsm'.Nc'.Nd').R.M.Nc'.Pam.Sidm)}
   _KeyGen(M.Nc'.Nd'.PRF(Pmsm'.Nc'.Nd'))) =|>
State':= 15
/\ Rcrdate':= new()
/\ SND_M({{Assr.Np.Sn.Rcrdate'.Issuer.({Assr.Np.Sn.Crdate.{R.Kr}
   _inv(Kca)}_(inv(Kp)))}_(inv(Kr)).{R.Kr}_inv(Kca)}
   _KeyGen(R.Pmsm'.Nc.Nd'))
/\ secret(Np,sec_np_rm,{R,M})
/\ request(M,R,auth_nc_nd,Nc.Nd)
/\ request(R,P,auth_reg_cert,{R.Kr}_inv(Kca))


15. State  = 15
/\ RCV_M({{Assr'.Np'.Sn'.{R.Kr'}_inv(Kca)}_(inv(Km))}
          _KeyGen(M.Pmsm'.Nc.Nd')) =|>
State':= 17 /\ SND_P({{Assr'.Np'.Sn'.{R.Kr'}_inv(Kca)}
         _(inv(Km))}_(KeyGen(R.Pms.Na.Nb)))
/\ witness(R,P,auth_reg_certchain,{R.Kr'}_inv(Kca))

end role

role masa (P, R, M  : agent,
Km, Kp, Kca  : public_key,
KeyGen, PRF    : hash_func,
Sn       : text, %serial Number
Issuer     : text,
Aki         : text,  %authorityKeyIdentifier
Sln      : text,  %subjectAltName
SND_R, RCV_R, SND_P, RCV_P   : channel (dy))
played_by M def=

local SeID, Assr , Np, Nc, Nd, Pmsm, Sidm, Pam : text,
Crdate, Rcrdate     : text,
State               : nat,
Kr              : public_key,
```

```
Finishedm: hash(hash(text.text.text).agent.agent.text.text.text),
ClientKm, ServerKm: hash(agent.text.text.hash(text.text.text))

const auth_sn, auth_np, auth_np1, auth_idev, auth_reg_certchain,
auth_reg_cert1,auth_nc_nd, auth_nc_nd2  : protocol_id%,

init State := 5

transition

1. State  = 5
/\ RCV_R(R.Nc'.Sidm'.Pam') =|>
State':= 7
/\ Nd' := new()
/\ SND_R(Nd'.Sidm'.Pam'.{M.Km}_inv(Kca))
/\ witness(M,R,auth_nc_nd,Nc'.Nd')

2. State  = 7
/\ RCV_R({Pmsm'}_(Km').{R.Kr'}_inv(Kca).{Finishedm'}_ClientKm')
/\ Finishedm = PRF(PRF(Pmsm'.Nc.Nd').R.M.Nc.Pam.Sidm)
/\ ClientKm = KeyGen(R.Nc.Nd'.PRF(Pmsm'.Nc.Nd'))=|>
State':= 19
/\ ServerKm' := KeyGen(M.Nc.Nd'.PRF(Pmsm'.Nc.Nd'))
/\ SND_R( {PRF(PRF(Pmsm'.Nc.Nd').R.M.Nc.Pam.Sidm)}
   _KeyGen(M.Nc.Nd'.PRF(Pmsm'.Nc.Nd')) )
/\ request(M,R,auth_nc_nd2,Nc.Nd)

3. State  = 19
/\ RCV_R({{Assr'.Np'.Sn.Rcrdate'.Issuer'.({Assr'.Np'.Sn.Crdate'.
      {R.Kr'}_inv(Kca)}_(inv(Kp')))}_(inv(Kr')).
      {R.Kr'}_inv(Kca)}_KeyGen(R.Pmsm'.Nc.Nd')) =|>
State':= 21
/\ SND_R({{Assr'.Np'.Sn.{R.Kr'}_inv(Kca)}_(inv(Km))}
         _KeyGen(M.Pmsm'.Nc.Nd'))
/\ witness(M,P,auth_np,Np')
/\ witness(M,P,auth_sn,Sn)
/\ request(M,P,auth_idev,Sn)


end role
```

```
role session(P, S, M        : agent,
Kp, Kr, Km, Kca : public_key,
KeyGen, PRF     : hash_func,
Sn              : text, %serial Number
Issuer          : text,
Aki             : text,  %authorityKeyIdentifier
Sln             : text  %subjectAltName
)
def=

local SP, SR, SM, RP, RR, RM  : channel (dy)

composition
pledge(P,S,M,Kp,Km,Kca,KeyGen,PRF,Sn,Issuer,Aki,Sln,SP,RP,SM,RM)
/\ registrar(P,S,M,Kr,Km,Kca,KeyGen,PRF,SR,RR,SM,RM)
/\ masa(P,S,M,Km,Kp,Kca,KeyGen,PRF,Sn,Issuer,Aki,Sln,SM,RM,SP,RP)

end role

role environment()
def=

const p,r,m            : agent,
kp, kr, km, kca, ki    : public_key,
keygen, prf            : hash_func,
sn                     : text, %serial Number
issuer                 : text,
aki                    : text,  %authorityKeyIdentifier
sln                    : text   %subjectAltName

intruder_knowledge = {p,r,m,kp,kr,m,kca,ki,inv(ki)
                      ,{i.ki}_inv(kca)  %%% 2nd session
%                     ,{i.ki}_inv(kca)  %%% 3rd session
}

composition
session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln)
%   /\  session(p,r,m,kp,kr,km,kca,keygen,prf,sn,issuer,aki,sln)
%   /\ session(p,i,m,kp,ki,km,kca,keygen,prf,sn,issuer,aki,sln)
%   /\ session(i,r,m,ki,kr,km,kca,keygen,prf,sn,issuer,aki,sln)
%   /\ session(p,r,i,kp,kr,ki,kca,keygen,prf,sn,issuer,aki,sln)
```

```
end role


goal

secrecy_of sec_np_pr,sec_np_rm %confidentiality of Np
secrecy_of tls_pledgek,tls_registrark,tls_registrarrmk,tls_masarmk
authentication_on auth_sn
authentication_on auth_na_nb
authentication_on auth_na_nb2
authentication_on auth_reg_cert
authentication_on auth_nc_nd
authentication_on auth_nc_nd2
authentication_on auth_idev  %Masa can authenticate pledge on serialNumber
authentication_on auth_reg_certchain
authentication_on auth_np  %Registrar authenticates voucher by checking
nonce Np.
authentication_on auth_np1 %Plegde authenticates voucher by checking nonce
Np.

end goal
```