CDVT Attack Detection Tool User Guide

1. Overview

The CDVT/AD tool is developed by the Data Communication Security Laboratory at the Department of Electronic and Computer Engineering, University of Limerick, Ireland. The development of this tool and its underlying techniques has been partially funded by:

- Science Foundation Ireland - Research Frontiers Programme 11/RFP.1/CMS 3340, Principal Investigators: Professor Tom Coffey, Dr Reiner Dojen
- Science Foundation Ireland - Research Frontiers Programme CMSF631, Principal Investigators: Professor Tom Coffey, Dr Reiner Dojen

For any queries regarding the CDVT/AD tool, please contact Reiner Dojen at reiner.dojen@ul.ie.

The latest version of the tool and of this user guide can be obtained at http://www.dcsl.ul.ie/CDVT/AD-tool.

2. Usability of CDVT/AD Tool

The verification tool has the capability of detecting protocol design weaknesses that can be exploited by replay or parallel session attacks (cf. Figure 1). The CDVT AD tool can be used for the following tasks:

- Detection of design weaknesses on security protocols that can be exploited by replay or parallel session attacks
- Finding the reasons for failure of verified protocols' goals (trace-ability of verification results) and the capability to perform an exhaustive search for different proof traces
- Helping with the re-design and correction of security protocols which are found to contain weaknesses, by revealing the weaknesses and the exact message and step where each weakness occurs in the protocols and further, by providing information/guidelines how to correct these weaknesses.

The rules of this tool are based upon the guidelines published by Jurcut, A., Coffey, T. and Dojen, R.: "Design Guidelines for Security Protocols to Prevent Replay & Parallel Session Attacks", Elsevier Computer & Security, Volume 45, September 2014, pp. 255-273

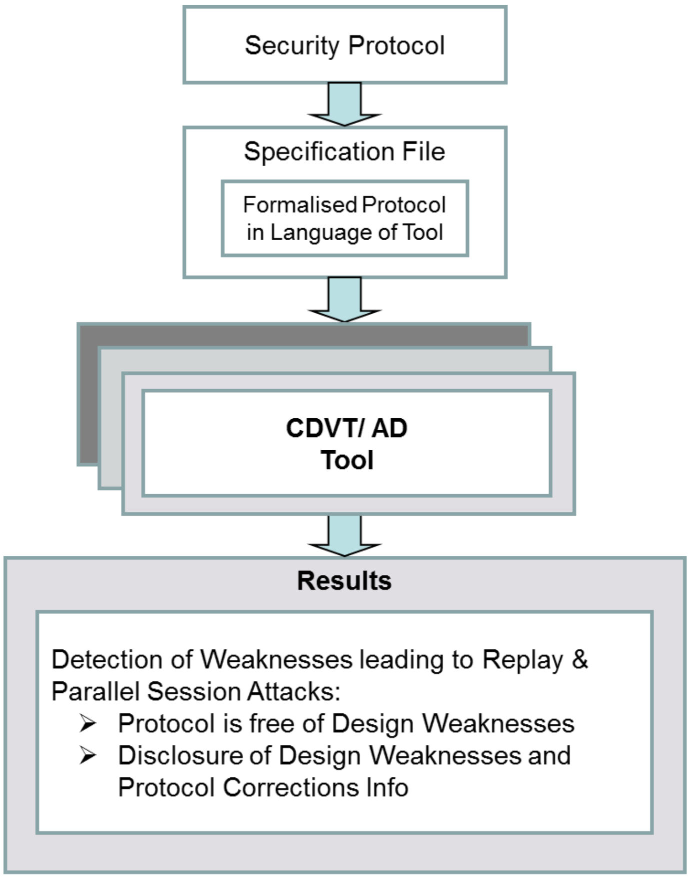

Security Protocol
Specification File
Formalised Protocol in Language of Tool
CDVT/ AD Tool
Results
Detection of Weaknesses leading to Replay & Parallel Session Attacks:
➢ Protocol is free of Design Weaknesses
➢ Disclosure of Design Weaknesses and Protocol Corrections Info

Fig. 1. The CDVT/AD Verification Tool

3. How to use CDVT/AD Tool?

Security protocols can be analysed with the CDVT/AD tool as follows:

1. A formal specification of the security protocol is stored in a text file. This specification must follow the language detailed in section 4.
2. Download and run the latest version of the CDVT/AD tool.
3. The user selects "Start Verification" button (see Figure 2) to select and read in the file containing the formal specification of protocol to be verified.
4. The CDVT performs the attack detection of protocol and displays the results of the analysis (see section 5 for information on how to interpret the tool's output)

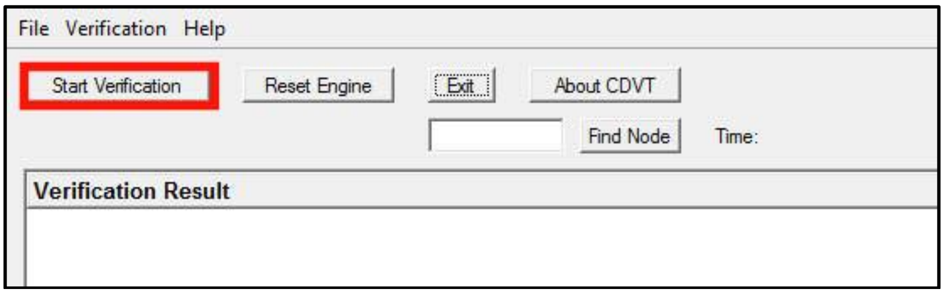


Fig. 2. CDVT/AD tool: Starting Attack Detection

4. Input Language

A protocol is specified declaratively in an individual text file (.txt), following well-defined syntax statements. The formalisation specification of a protocol incorporates two parts: Assumptions and Steps. General form of specifications is as following:

Label : *Statement* ;

Label is An and Sn for Assumptions and Steps, respectively, 'n' is a positive integer (e.g. protocol assumptions defined with label A1, A2, etc., protocol steps defined with S1, S2).

Statement can have one of the following structures

- *Principal Operator Time Data;*
- *Principal Operator Time Statement;*
- *Principal ComOperator Principal Time Data*
- *Principal know Time NOT (Statement);*
- *Principal believe Time NOT (Statement);*
- (*Statement*);
- (*Statement*) AND (*Statement*);
- (*Statement*) OR (*Statement*);
- (*Statement*) IMPLY (*Statement*)

where *Principal* means either "an agent" or "trusted third party TTP" that follows the data format of a principal or trusted principal detailed below.

Data components can be any of the following:

- Symmetric keys start with "K" followed by a single lower case letter followed by any combination of letters (any case), digits and the underscore (e.g. Kab);
- Public keys start with "K" followed by a single lower case letter followed by any combination of letters (any case), digits and the underscore and ending in string "Pub" (e.g. KaPub);
- Private keys start with "K" followed by a single lower case letter followed by any combination of letters (any case), digits and the underscore and ending in string "Priv" (e.g. KaPriv);
- Nonces start with "N" followed by a single lower case letter followed by any combination of letters (any case), digits and the underscore(e.g. Na);
- Timestamps start with "TS" followed by a single lower case letter followed by any combination of letters (any case), digits and the underscore (e.g. TSa);
- Generic (binary) data components start with a lower case letter followed by any combination of letters (any case), digits and the underscore;
- Functions start with "F" followed by any combination of letters (any case), digits and the underscore(e.g. Fincrement(Na));
- Hash functions start with "H" followed by any combination of letters (any case), digits and the underscore (e.g. H(x));
- Key material data components are denoted by "KMaterial()" where the data components are inside the brackets (e.g. KMaterial(x));
- Shared secret data components are denoted by "SharedWith()" where the data components consisting of shared secret component and the second principal sharing the component are inside the brackets e.g. "SharedWith(x,B)" indicates that component x is a secret shared with principal B. The following expression indicates that principals A and B share, at time t0, the shared secret component x: A know at[0] SharedWith(x,B);
- Encryptions are denoted by curly brackets ("{}") where the encrypted data is inside the brackets and the key used for encryption follows the brackets, e.g. "{Na}Kab" indicates

the nonce “Na” encrypted under key “Kab”; Note: {{x}K}K = x, for encryption with symmetric keys, and {{x}KPub}KPriv = x = {{x}KPriv}KPub, for encryption with public/private keys.
- Trusted principals start with “TTP” followed by any combination of letters (any case), digits and the underscore;
- Wildcard principal “Zero” indicates any principal other than the principal in the governing context;
- Principals are all strings starting with a capital letter not already covered, followed by any sequence of characters and letters.

Operator can be any of the following:
- know
- believe
- possess

ComOperator can be any of the following:
- sendto
- receivefrom

Time is defined as “at [*index*]” where index is a number:
- -1: indicates previous protocol runs;
- 0: indicates the beginning of the current protocol run;
- 1 … n: indicates the time at a specific step between 1 and n.

Comments are inserted in C++ style, i.e. single line comments only, starting from double forward slash (‘//’) until end of line.

For example, a step of a protocol: S1: A -> B: A,{Na}Kab can be formalised as:

S1: B receivefrom A at[1] A,{Na}Kab;

The following statements expresses that the nonce Na is fresh for principal A:

A1: A know at[0] NOT(Zero possess at[0] Na);

It reads “A knows at time t0 (the beginning of the protocol) that no other principal possessed nonce Na at time t0”.

The following expression states that principals A shares the component x with B.

A know at[0] SharedWith(x,B);

It reads “A knows at time t0 it shares secret x with B. If the sharing property is important to both parties, two statements are required:

A know at[0] SharedWith(x,B);
B know at[0] SharedWith(x,A);

5. Outputs of the CDVT/AD Tool

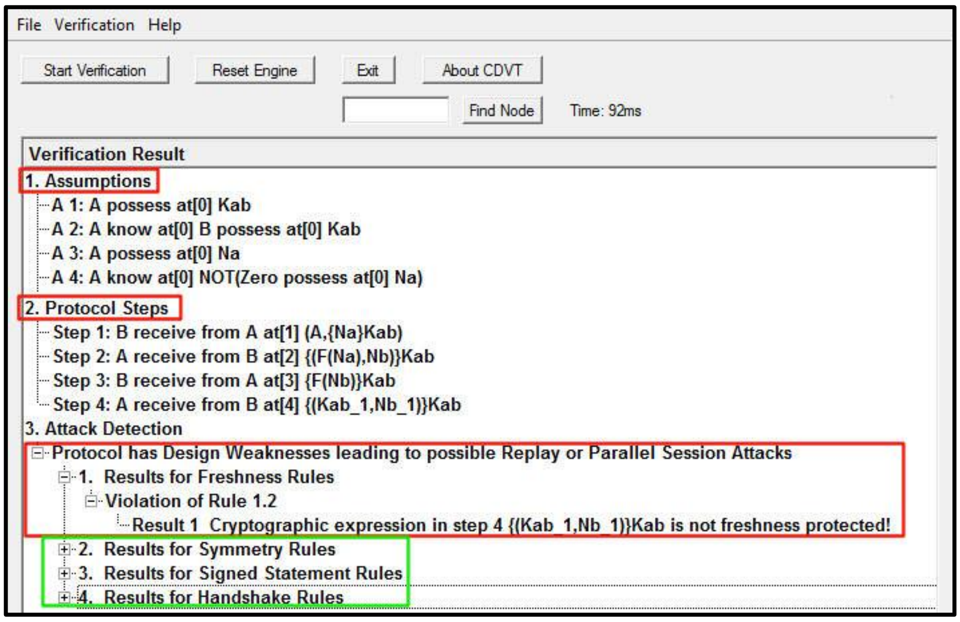
The output sections outlined in Table 1 are displayed.

Output	Description
Assumptions	Displays the initial assumptions specified for the verified protocol.
Protocol Steps	Displays the formalised protocol steps

	specified for the verified protocol.
Attack Detection	Displays the results of the attack detection process.

Table 1. Output Sections of the CDVT/AD Tool

All outputs, i.e. Assumptions, Steps, Attack Detection, can be expanded by double clicking on them (see Fig. 3 – extended output is marked in red and the output that is still extendible is marked in green).



```
Verification Result
1. Assumptions
   A 1: A possess at[0] Kab
   A 2: A know at[0] B possess at[0] Kab
   A 3: A possess at[0] Na
   A 4: A know at[0] NOT(Zero possess at[0] Na)
2. Protocol Steps
   Step 1: B receive from A at[1] (A,{Na}Kab)
   Step 2: A receive from B at[2] {(F(Na),Nb)}Kab
   Step 3: B receive from A at[3] {F(Nb)}Kab
   Step 4: A receive from B at[4] {(Kab_1,Nb_1)}Kab
3. Attack Detection
   Protocol has Design Weaknesses leading to possible Replay or Parallel Session Attacks
      1.  Results for Freshness Rules
         Violation of Rule 1.2
            Result 1  Cryptographic expression in step 4 {(Kab_1,Nb_1)}Kab is not freshness protected!
      2.  Results for Symmetry Rules
      3.  Results for Signed Statement Rules
      4.  Results for Handshake Rules
```

Fig. 3. The CDVT/AD Tool extendable Outputs

5.1 Attack Detection Output (*Output for Attack Detection Logic*)

The "***Attack Detection***" output can reveal one of the following two results:

- "*Protocol is Free of Design Weaknesses leading to possible Replay or Parallel Session Attacks*" – if none of the Attack Detection Logic *rules* is triggered. Therefore, protocol is considered free of mountable replay and parallel session attacks.
- "*Protocol has Design Weaknesses leading to possible Replay or Parallel Session Attacks*" – if at least one of the Attack Detection Logic *rules* is triggered (i.e. the prerequisites of at least one of the detection rules can be derived from the formalised protocol specification). The investigations of the protocol weaknesses inform/guide the user how to correct and re-designed the faulty protocol.

CDVT/AD tool incorporates a set of Attack Detection *rules* that are classified into four main categories:

(1) Freshness Rules,
(2) Symmetry Rules,
(3) Signed Statements Rules,
(4) Handshake Rules.

The result of "***Attack Detection"*** output can be expanded to inspect the categories of rules triggered. Further, each of the four categories of rules can be expanded to reveal the rules triggered in that category. Furthermore each rule triggered can be expanded to disclose the details of the detected weakness and how this weakness can be corrected. If none of the detection rules in a category are triggered the tool outputs: *"No weaknesses found"* for the respective category. The output for each Attack Detection Logic rule triggered is given in Tables 2-1 and 2-2.

Categories of Rules	Rules Violation	Described Weakness
Results for Freshness Rules	Violation of Rule R1.1	Cryptographic expression in step r: {x}k does not contain freshness identifier!
	Violation of Rule R1.2	Cryptographic expression in step r: {x}k is not freshness protected!
	Violation of Rule R1.3	Cryptographic expression in step r: {x}k is not freshness protected!
Results for Symmetry Rules	Violation of Rule R2.1	The pair of cryptographic expressions: {x}k1 from step q and {y}k2 from step r are symmetric in opposite directions!
	Violation of Rule R2.2	The pair of cryptographic expressions: {x}k1 from step q and {y}k2 from step r are symmetric in opposite directions!
	Violation of Rule R2.3	The pair of cryptographic expressions: {x}k1 from step q and {y}k2 from step r are symmetric in opposite directions!
	Violation of Rule R2.4	The pair of cryptographic expressions: {x}k1 from step q and {y}k2 from step r are symmetric in opposite directions!
Results for Signed Statement Rules	Violation of Rule R3.1	The signed statement in step r: {x}KAPriv is not receiver bound ! Its receiver identity B should be included!
	Violation of Rule R3.2	The signed statement in step r: {x}KAPriv is not receiver bound ! Its receiver identity B should be included!
	Violation of Rule R3.3	The signed statement in step r: {x}KAPriv is not receiver bound ! Its receiver identity B should be included!
	Violation of Rule R3.4	The cryptographic expression in step r: {x}k should contain the owner of public key identity A!

Table 2-1. CDVT/AD Tool Output for Attack Detection Rules (Part 1)

Categories of Rules	Rules Violation	Described Weakness
Results for Signed Statement Rules	Violation of Rule R3.1	The signed statement in step r: {x}K_{APriv} is not receiver bound ! Its receiver identity B should be included!
	Violation of Rule R3.2	The signed statement in step r: {x}K_{APriv} is not receiver bound ! Its receiver identity B should be included!
	Violation of Rule R3.3	The signed statement in step r: {x}K_{APriv} is not receiver bound ! Its receiver identity B should be included!
	Violation of Rule R3.4	The cryptographic expression in step r: {x}k should contain the owner of public key identity A!
Results for Handshake Rules	Violation of Rule R4.1	At least one of the cryptographic expressions in step r: {x}K_{AB}, e.t.c. should be strong sender bound. Its sender identity A should be included!
	Violation of Rule R4.2	At least one of the cryptographic expressions in step r: {x}K_{AB}, e.t.c. should be strong sender bound. Its sender identity A should be included!
	Violation of Rule R4.3	At least one of the cryptographic expressions in step r: {x}K_{AB}, e.t.c. should be strong sender bound. Its sender identity A should be included!
	Violation of Rule R4.4	At least one of the cryptographic expressions in step r: {x}$K_{APriv,}$, e.t.c. should be receiver bound. Its receiver identity B should be included!
	Violation of Rule R4.5	At least one of the cryptographic expressions in step r: {x}$K_{APriv,}$, e.t.c. should be receiver bound. Its receiver identity B should be included!
	Violation of Rule R4.6	At least one of the cryptographic expressions in step r: {x}K_{BPub}, e.t.c. should be sender bound. Its sender identity A should be included!
	Violation of Rule R4.7	At least one of the cryptographic expressions in step r: {x}K_{BPub}, e.t.c. should be sender bound. Its sender identity A should be included!

Table 2-2. CDVT/AD Tool Output for Attack Detection Rules (Part 2)

6. Sample Protocols Formal Specs & Verification

In this section, sample protocols are formalised and analysed.

6.1. Andrew Secure RPC Protocol

The Andrew Secure RPC protocol was proposed for the distribution of a fresh shared key between two principals, who already share a key:

1. A → B : A, {Na}Kab
2. B → A : {succNa, Nb}Kab
3. A → B : {succNb}Kab
4. B → A : { Kab_1, Nb_1}Kab

6.1.1. Formalisation of Protocol

Initial assumptions are statements defining what each principal possesses and knows at the beginning of a protocol run. The following specifies the initial assumptions of the Andrew Secure RPC protocol:

A1: A possess at[0] Kab;
A2: A know at[0] B possess at[0] Kab;
A3: A possess at[0] Na;
A4: A know at[0] NOT(Zero possess at[0] Na);
A5: B possess at[0] Kab;
A6: B know at[0] A possess at[0] Kab;
A7: B possess at[0] Nb;
A8: B know at[0] NOT(Zero possess at[0] Nb);
A9: B possess at[0] Nb_1;
A10: B know at[0] NOT(Zero possess at[0] Nb_1);
A11: B know at[0] KMaterial(Kab_1);

Statements A1-A4 define the initial assumptions for principal A before a protocol run with principal B (i.e. at time t0), as follows: Assumption A1 states that A possesses symmetric key Kab. A2 specifies that A is aware of the fact that principal B possesses the key Kab. A3 specifies that A possesses the nonce Na and assumption A4 states that A knows that nonce Na is fresh for the current run of the protocol. Statements A5-A11 expresses the initial assumptions stating B's possessions and knowledge before the start of the protocol run. A5 states that B possesses symmetric key Kab. A6 specifies that B is aware of the fact that principal A possesses the key Kab. A7 and A9 express the fact that B possesses nonces Nb and Nb_1. A8 and A10 state that B knows that nonces Nb and Nb_1 are fresh for the current run of the protocol. A11 expresses the fact that the component Kab_1 is to be used in the generation of a new key.

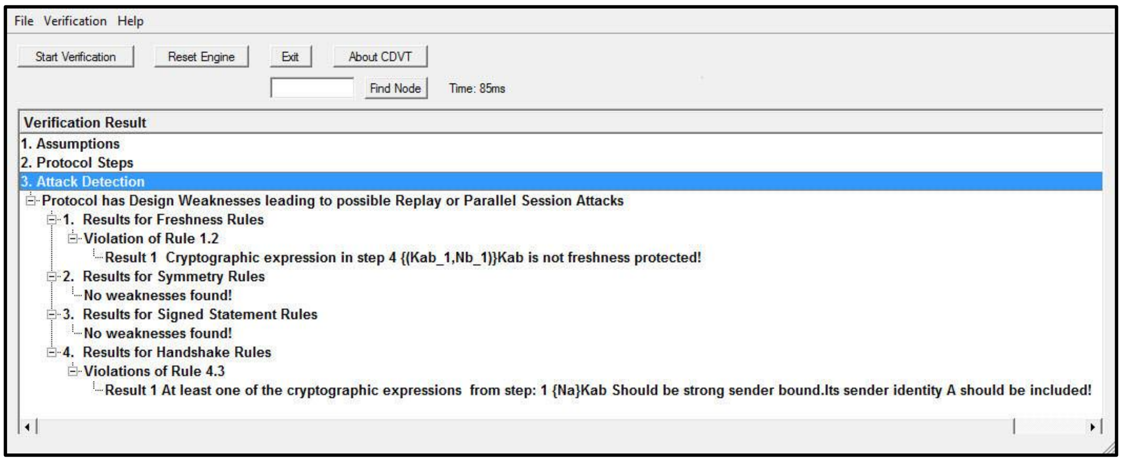
The Andrew Secure RPC protocol steps are formalised as follows:

S1: B receivefrom A at[1] A,{Na}Kab;
S2: A receivefrom B at[2] {(F(Na), Nb)}Kab;
S3: B receivefrom A at[3] {F(Nb)}Kab;
S4: A receivefrom B at[4] {(Kab_1, Nb_1)}Kab;

Statement S1 states that principal B receives from principal A in step one (i.e. at time t1) a request, which contains A's identity and a nonce, Na, encrypted with their shared key Kab. S2 states that A receives from B in step two (i.e. at time t2) a successor to its nonce Na (formalised using the generic function F(Na)) along with a new nonce, Nb, generated by B. This message is also encrypted with the shared key Kab. S3 states that B receives from A in step three (i.e. at time t3) the successor to B's nonce, F(Nb) encrypted with their shared key Kab. S4 expresses that A receives from B in step four (i.e. at time t4) a new session key Kab_1 along with a new generated nonce Nb_1, all encrypted with key Kab.

6.1.2 Attack Detection Results

Two types of detection rules: a freshness rule R1.2 and a handshake rule R4.3 are triggered (see Fig. 3). Therefore, two weaknesses in the design of Andrew Secure RPC protocol, identifying a replay and a parallel session vulnerability are revealed. First weakness revealed is the fact that the cryptographic expression in step 4 of the Andrew Secure RPC protocol {(Kab_1, Nb_1)}Kab, is not freshness protected (freshness rule R1.2 triggered). This weakness can lead to a replay attack and allow an intruder to substitute a previously recorded message 4 (from B to A) and mislead principal A to accept an old and possibly compromised session key Kab_1. The second weakness (rule R4.3 triggered) is caused by the faulty construction of the SOSH handshake of the protocol, consisting of the first two steps S1 and S2. The weakness is the fact that none of the cryptographic expressions exchanged in S1 or S2 of this challenge-response is strong sender bound or strong receiver bound. This weakness can be exploited by a parallel session attack enabling an intruder to masquerade as a legitimate principal, since is impossible to tell whether the cryptographic messages transmitted in S1 and S2 are intended to be sent by A to B and/or vice versa. Hence, the outcome of the tool describes a solution to fix this problem, by recommending to add the identity of the sender A in the content of the cryptographic expression {Na}Kab, transmitted in step 1 of the protocol (i.e. modify the content of the cryptographic expression {Na}Kab in order to become strong sender bound).



```
Verification Result
1. Assumptions
2. Protocol Steps
3. Attack Detection
Protocol has Design Weaknesses leading to possible Replay or Parallel Session Attacks
    1. Results for Freshness Rules
        Violation of Rule 1.2
            Result 1  Cryptographic expression in step 4 {(Kab_1,Nb_1)}Kab is not freshness protected!
    2. Results for Symmetry Rules
        No weaknesses found!
    3. Results for Signed Statement Rules
        No weaknesses found!
    4. Results for Handshake Rules
        Violations of Rule 4.3
            Result 1 At least one of the cryptographic expressions  from step: 1 {Na}Kab Should be strong sender bound.Its sender identity A should be included!
```

Fig. 4. Attack Detection outcome for Andrew Secure RPC

6.2 LKY nonce-based Mutual Authentication Scheme

Authentication session of LKY scheme:

1. U_i ->AS: $M_1 = (ID_i,C_2)$
2. AS->U_i: $M_2=(V_1,C_3)=(h(C_2,N_1), h(ID_i \oplus x) \oplus N_2)$
3. U_i ->AS: $M_3 = (V_2) = h(C_3,N_2)$

6.2.1 Formalisation of Protocol

Initial assumptions:

A1: A possess at[0] H({A}datax);
A2: A know at[0] TTP possess at[0] H({A}datax);
A3: A possess at[0] Na;
A4: A know at[0] NOT(Zero possess at[0] Na);
A5: TTP possess at[0] H({A}datax);
A6: TTP know at[0] A possess at[0] H({A}datax);

A7: TTP possess at[0] Nttp;
A8: TTP know at[0] NOT(Zero possess at[0] Nttp);

Statements A1-A4 define the initial assumptions for principal A before a protocol run with TTP (i.e. at time t0). Assumption A1 states that A possesses symmetric key "H({A}datax)". A2 specifies that A is aware of the fact that TTP possesses "H({A}datax)". A3 specifies that A possesses the nonce Na and assumption A4 states that A knows that nonce Na is fresh for the current run of the protocol. Statements A5-A8 define the initial assumptions of TTP's possessions and knowledge before the start of the protocol run. A5 states that TTP possesses key "H({A}datax)". A6 specifies that TTP is aware of the fact that principal A possesses "H({A}datax)". A7 expresses the fact that TTP possesses the nonce Nttp and A8 states that TTP knows that Nttp is fresh for the current run of the protocol.

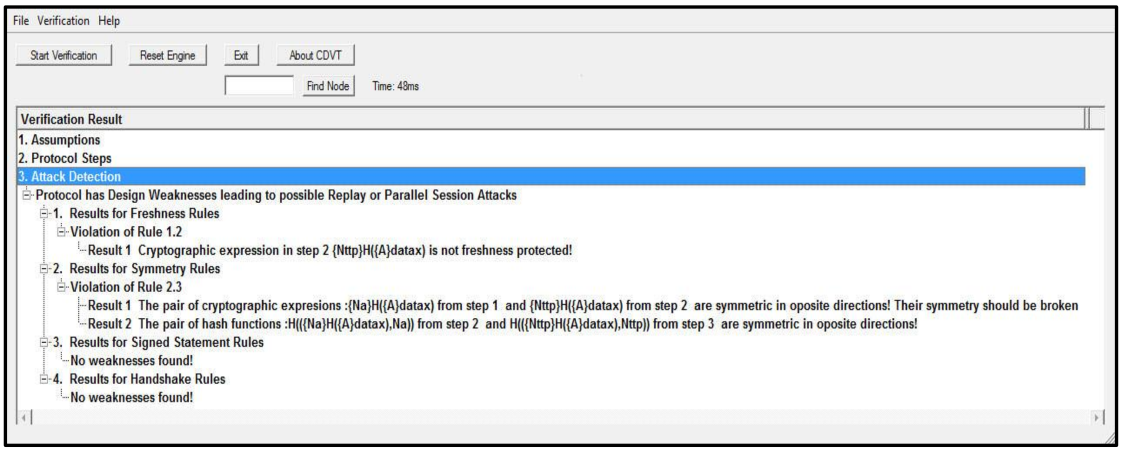
The steps are formalized as follows:

S1: TTP receivefrom A at [1] A,{Na}H({A}datax);
S2: A receivefrom TTP at [2] H({Na}H({A}datax),Na), {Nttp}H({A}datax);
S3: TTP receivefrom A at [3] H({Nttp}H({A}datax), Nttp);

6.2.2 Attack Detection Results

Three weaknesses in the design of LKY scheme, identifying freshness and parallel session vulnerabilities are revealed (cf. Figure 5). Hence, two attack detection rules of the verification tool (the freshness rule R1.2 - triggered once) and the symmetry rule R2.3 - triggered twice) are triggered. The result obtained with respect to the freshness rules is that the cryptographic expression in step 2 {Nttp}H({A}datax) is not freshness protected. This implies that {Nttp}H({A}datax) does not contain data which the receiver A recognizes as being fresh (i.e. a nonce previously generated by A in the same protocol run). The results derived for the symmetry rules reveal the following two weaknesses in the LKY scheme:

- the cryptographic expressions {Na}H ({A}datax) of step 1 and {Nttp}H({A}datax) of step 2 are symmetric;
- the hashed expressions H({Na}H({A}datax), Na) of step 2 and H({Nttp}H({A}datax,A), Nttp) of step 3 are symmetric.



```
Verification Result
1. Assumptions
2. Protocol Steps
3. Attack Detection
 Protocol has Design Weaknesses leading to possible Replay or Parallel Session Attacks
   1.  Results for Freshness Rules
     Violation of Rule 1.2
       Result 1  Cryptographic expression in step 2 {Nttp}H({A}datax) is not freshness protected!
   2.  Results for Symmetry Rules
     Violation of Rule 2.3
       Result 1  The pair of cryptographic expresions :{Na}H({A}datax) from step 1  and {Nttp}H({A}datax) from step 2  are symmetric in oposite directions! Their symmetry should be broken
       Result 2  The pair of hash functions :H(({Na}H({A}datax),Na)) from step 2  and H(({Nttp}H({A}datax),Nttp)) from step 3  are symmetric in oposite directions!
   3.  Results for Signed Statement Rules
     No weaknesses found!
   4.  Results for Handshake Rules
     No weaknesses found!
```

Fig. 5. Attack Detection Verification outcome for LKY scheme