Automated Logic-Based Technique for Formal Verification of Security Protocols

Anca D. Jurcut

***Abstract*—The design of secure protocols is complex and prone to error. Formal verification is an imperative step in the design of security protocols and provides a rigid and thorough means of evaluating the correctness of security protocols. This paper discusses the process of formal verification using a logic-based technique for detecting protocol weaknesses that are exploitable by freshness and interleaving attacks. This technique is realised as a special purpose logic for attack detection that can be used throughout the design stage, i.e. it subjects a draft of a protocol to formal analysis prior to its publication or deployment. For any detected failure the analysis will also reveal reasons for the weaknesses, facilitating design corrections. A summary of the attack detection logic is presented and its ability to detect weaknesses is demonstrated by applying it to a smart-card based authentication protocol. Further, a prototype implementation of the attack detection logic theory is introduced. An empirical study is presented that assesses the effectiveness and efficiency of the proposed automated technique by applying it to a set of protocols, incorporating some with known vulnerabilities and some that are known to be secure. This study confirms the ability of the technique to detect all design weaknesses. Additionally, it establishes the efficiency of the verification technique, in terms of memory requirements (study was carried out on a computing platform of 2GB of RAM) and execution times (milliseconds) required for protocol verification.**

***Index Terms*—Attacks, formal verification, logic-based verification tool security protocols.**

I. Introduction

The security of electronic networks and information systems is a critical issue for the use of new technologies in many fields of life. The massive growth in communications, in particular in the wireless sector and internet of things, causes an ever changing environment for today's communication services. Security protocols are required to ensure the security of both the communications infrastructure itself and the information that runs through it. Designing error-free security protocols that are impervious to attack techniques, such as freshness and interleaving sessions (i.e. impersonation attacks, man-in-the-middle attacks, oracle attacks, multiplicity attacks and other types of parallel session attacks) is an extremely challenging task [1]. The challenge comes mainly from the difficulty of foreseeing all possible operative scenarios of an attacker, which can include concurrent execution of several protocol sessions and various different attack strategies. For example, in a freshness attack the adversary uses components of the messages from previous runs of the protocol to gain an advantage, while in an interleaving session attack the adversary uses multiple runs of the protocol to gather knowledge. Even though protocols' structure sometimes can be simple (only a few messages are exchanged, in general, among a reduced number of protocol principals), they are prone to errors that are subtle and very hard to detect manually.

Formal verification of a security protocol is a critical part of the design process of security protocols [2], as it provides a systematic way to detect design flaws. The importance of formal methods has been practically demonstrated by several success stories of protocol weaknesses discovered using formal analysis methods - often several years after the original publication [3]-[15]. As a consequence, when designing a protocol, it needs to be formally verified in order to prove that it meets its security goals and is free of weaknesses that might be exploitable by mountable attacks. Thus, it improves confidence in the security of the designed protocol.

Existing formal methods include two main different approaches to the verification problem: logic-based analysis [3]-[18] normally used to verify the correctness of security goals of a protocol and brute force methods of state-space exploration [19], [20], respectively used for the detection of attacks against a protocol. While both approaches have been successfully employed to detect weaknesses in security protocols, currently there is a significant need for techniques which can achieve both objectives of formal security protocol verification: proving that the verified protocol meets its security goals and demonstrating the absence of mountable attacks against the protocol. Further, the automation of the verification process minimizes the risk of faulty proofs and simplifies the verification process for the protocol verifier. In addition, logics have an advantage in that they are usually decidable and often efficiently computable and thus can be completely automated [17].

This paper is concerned with the use of logic techniques for the formal analysis of security protocols. We demonstrate the application of a recently proposed technique, which was realised as a special purpose logic for attack detection [21]. The Attack Detection Logic expands the capabilities of logic-based verification techniques, by adding attack detection to their traditional role of proving that protocols meet their security goals. We demonstrate the ability of the logic to detect attacks, by presenting in detail the analysis process, when manually applying this technique to a smart-card authentication protocol [9]. For any detected failure, the analysis will also reveal reasons for the weaknesses, facilitating design corrections of the protocol verified. Further, we present a prototype implementation of the Attack Detection Logic, which was integrated into an

Manuscript received September 10, 2018; revised November 15, 2018.

A. D. Jurcut is with the School of Computer Science, University College Dublin, Ireland (e-mail: anca.jurcut@ucd.ie).

doi: 10.18178/jacn.2018.6.2.258

existing logic-based verification tool CDVT [22], using a modal logic of knowledge and belief. Hence, we show that our proposed automated logic can cooperate with conventional verification logics. Together the combined logics have the capability to achieve both objectives of formal security protocol verification (i.e. proving that the verified protocol meets its security goals and demonstrating the absence of mountable attacks against the protocol). Empirical results on verifying a range of security protocols using the automated prototype implementation of the Attack Detection Logic are also presented. Successful detection of all attacks shows the effectiveness of the proposed automated logic. Furthermore, the fast execution times demonstrate the efficiency of this technique.

The remainder of this paper has the following structure: Section 2 gives an overview of Attack Detection Logic and presents a detailed explanation on how to apply the logic. A prototype implementation of the attack detection logic theory is introduced in Section 3. Section 4 shows an empirical study that assesses the effectiveness and efficiency of the proposed automated technique. Finally, Section 5 concludes the paper. Additionally, a sample verification of the protocol [9], using the proposed automated technique is demonstrated in the Appendix.

II. Logic-Based Technique for Formal Verification of Security Protocols

In 2017, Jurcut, Coffey and Dojen [21] proposed a novel logic with attack detection capabilities for the formal verification of cryptographic security protocols. This logic, referred to as the Attack Detection Logic, expands the capabilities of existing logic-based verification techniques, by adding attack detection to their traditional role of proving that protocols meet their security goals. The Attack Detection Logic is designed to detect security protocol weaknesses that can be exploited by freshness and interleaving session attacks (including identity attacks, man-in-the-middle attacks, unknown key-share attacks, oracle attacks, multiplicity attacks and other parallel session attacks).

The Attack Detection Logic has a unique attack detection capability for freshness and interleaving session attacks and it provides generic structures of the detected attacks. The proposed logic characterizes the general circumstances under which attacks may exist by examining the structure of message exchanges in a protocol. This examination takes into account:

1) Knowledge of the principals involved.
2) Role of the messages in the protocol.
3) The way messages are transmitted.
4) Content of messages.

The Attack Detection Logic is based on the set of protocol design guidelines introduced by Jurcut *et al.* in [1]. These guidelines are general purpose so as to encompass a wide variety of protocols and to address the following protocol message exchange situations:

1) Guidelines to Ensure Message Freshness covering:
- Freshness requirements with and without synchronized clocks.
- Transmission of components used in key generation.

2) Guidelines to Prevent Message Symmetry covering:
- Direct and indirect (via a TTP) exchanges of cryptographic transformations.

3) Guidelines for Signed Messages covering:
- Signed messages & parent cryptographic expressions.
- Signed messages contained by parent cryptographic expressions encrypted with symmetric or public keys.
- Signed messages intended for public key distribution.

4) Guidelines for Handshakes Construction covering:
- Direct and indirect POSH (“Public Out Secret In”), SOPH (“Secret Out Public In”), and SOSH (“Secret Out Secret In”) types of challenge-response handshakes using symmetric and asymmetric encryption.

The logic characterizes the general circumstances under which a potential attack may exist, by examining the protocol design and defines a logical formula that describes these circumstances. It consists of a language, sets of predicates, axioms, rules and semantics:

- The language introduces syntactic rules for building well-formed formulas of the logic.
- The predicates evaluate properties of message exchanges and their components as well as principals.
- The axioms enable reasoning about message characteristics in cryptographic protocols.
- The rules combine axioms to describe the circumstances in which a protocol is vulnerable to replay or parallel session attacks.
- The semantics ascribe meaning to the components of the logic theory (i.e. logical connectives and of the predicates).

A. Applying the Attack Detection Logic

In this section we show how our proposed technique can be used in the design/re-design process of security protocols. For this demonstration we:

- Manually apply the proposed Attack Detection Logic to a security protocol with known weaknesses.
- Show which of the attack detection rules are violated.
- Re-design the protocol according to the design guidelines and re-evaluate the amended protocol to ensure it is free of design weaknesses exploitable by freshness or interleaving attacks.

Evaluating a Protocol with Known Weaknesses. In this evaluation we manually analyse the nonce-based mutual authentication scheme of Lee, Kim and Yoo [9], which has known weaknesses [10]-[31]. This analysis is realised by formalising the protocol in the logic’s language and then using deductive reasoning to prove the presence of weaknesses exploitable by replay or parallel session attacks. During this deductive reasoning process all detection rules need to be applied to the protocol. However, the presented analysis only shows application of rules violated by the protocol.

1) Formalization of protocol

The authentication session of the LKY scheme [9] (E3(P)) is formalized as follows:

$$E_3(P)\begin{cases} S_1 : A \rightarrow TTP : A, \{N_A\}H(\{A\}datax) \\ S_2 : TTP \rightarrow A : H(\{N_A\}H(\{A\}datax), N_A), \{N_{TTP}\}H(\{A\}datax) \\ S_3 : A \rightarrow TTP : H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}) \end{cases}$$

The following initial assumptions are considered when manually applying the Attack Detection Logic:

A1: $P(A, H(\{A\}datax), S_0)$ - Principal A possesses data H({A}datax) which plays the role of a symmetric key shared with the system TTP, before the authentication session of the protocol (i.e. at time t0);

A2: $Gen(A, N_A, S_1)$ - Principal A generated NA in step S1;

A3: $P(TTP, H(\{A\}datax), S_0)$ - Trusted third party TTP possesses data H({A}datax) which plays the role of a symmetric key shared with the user A, before the authentication session of the protocol (i.e. at time t0);

A4: $Gen(TTP, N_{TTP}, S_2)$ - Trusted third party TTP generated nonce NTTP in step S2.

2) Applying the detection rules

Axioms (A2) of the logic states that a component x is fresh for recipient ($\Gamma(R,x)$) if x is a timestamp or a counter generated by the sender of x under the assumption of synchronized clocks and the recipient R can check this timestamp for timeliness or if R is the receiver of response step Sp that contains x and x is a function of a component wR freshly generated and sent by R in a previous initiation step of the same protocol run.

$$\begin{aligned} &(A2)\ \Gamma(R,x) \leftrightarrow \\ &(\exists Sr \in P : C(m(Sr), x) \wedge s(Sr) = G \wedge r(Sr) = R \wedge \\ &x \in T \wedge Gen(G, x, Sr) \wedge Fresh(x)) \vee \\ &(r(Sr) = R \wedge \exists Sp \in RS(P) : C(m(Sp), x) \wedge x = F(w_R) \wedge \\ &Fresh(w_R) \wedge (\exists So \in IS(P) : o < p \wedge Gen(R, w_R, So))) \end{aligned}$$

As none of the components in the cryptographic expression $\{N_{TTP}\}H(\{A\}datax)$ transmitted in response step S2 are timestamps or freshly generated by the recipient A, application of axiom (A2) yields:

$$\neg\Gamma(A, \{N_{TTP}\}H(\{A\}datax)) \quad (1)$$

Axiom (A4) defines a cryptographic expression {x}k as freshness protected ($\Phi(\{x\}k)$) if {x}k contains a fresh component for recipient or if there exists a step Sp in En(P) where {x}k is concatenated with a hashed expression H(y*), where y* contains a secret shared between the two principals G and R and y* contains a fresh component for recipient R of Sp. Further, G possesses y* and y* contains {x}k.

$$\begin{aligned} &(A4)\ \Phi(\{x\}k) \leftrightarrow \\ &(\exists z : C(\{x\}k, z) \wedge \Gamma(R, z)) \vee (\exists Sp \in En(P) : C(m(Sp), (\{x\}k, H(y*))) \\ &\wedge (\exists w : C(y*, w) \wedge \Theta(G, R, w) \wedge P(G, y*) \wedge \Gamma(R, y*) \wedge C(y*, \{x\}k))) \end{aligned}$$

As by (1) cryptographic expression is neither fresh for recipient nor part of a hashed expression containing a fresh component for recipient of step S2 (user A), application of *(A4)* establishes:

$$\neg\Phi(\{N_{TTP}\}H(\{A\}datax)) \quad (2)$$

As the cryptographic expression $\{N_{TTP}\}H(\{A\}datax)$ is the only cryptographic expression in S2 and as S2 is a response step, the following is derived:

$$S_2 \in E_3(P) \wedge S_2 \in RS(P) \wedge \forall\{x\}k \in m(S_2) : \neg\Phi(\{x\}k) \quad (3)$$

Rule *(R1.2)* states that if no cryptographic expression in a response step Sp of message exchange En(P) is freshness protected, then a replay attack can be mounted on that message exchange:

(R1.2) $\exists Sp \in En(P) : Sp \in RS(P) \wedge \forall\{x\}k \in m(Sp) : \neg\Phi(\{x\}k)$
$\rightarrow$ ReplayAttack $En(P)$)

According to (3) the prerequisites of attack detection rule *(R1.2)* are fulfilled and therefore a *replay attack (R) can be mounted* on E_3(P).

Continuing the analysis, the remaining rules are applied.

By definition of predicate *Symmetric(x,y)* the following pair of hash functions are symmetric:

$$Symmetric(H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP})) \quad (4)$$

Axiom *(A5)* states that two keys k1 and k2 are *matching keys* ($M\kappa(k1, k2)$) if either:

- keys k1, k2 are both symmetric keys and have the same value,
- keys k1, k2 are both symmetric keys and are shared with the same TTP,
- keys k1,k2 are both public keys,
- keys k1,k2 are both private keys.

$$\begin{aligned} &(A5)\ M\kappa(k1, k2) \leftrightarrow \\ &(k1 \in SymK \wedge k2 \in SymK \wedge (k1 = k2 \vee \exists G, R, TTP \in ENT : P(G, k1) \wedge \\ &P(R, k2) \wedge P(TTP, k1) \wedge P(TTP, k2)) \vee (k1 \in PubK \wedge k2 \in PubK) \vee \\ &(k1 \in PrivK \wedge k2 \in PrivK) \end{aligned}$$

Axiom *(A6)* defines a pair of cryptographic expressions {x}k1, {y}k2 as being *symmetric* if x, y are symmetric and keys k1, k2 are matching keys.

(A6) $Symmetric(\{x\}k1, \{y\}k2) \leftrightarrow Symmetric(x, y) \wedge M\kappa(k1, k2)$

Applying axiom *(A6)* to the cryptographic expressions of E_3(P) reveals:

$$Symmetric(\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax)) \quad (5)$$

The axiom *(A7)* defines two components, x and y as *principal value type equivalent* ($Pvte(x, y)$) if for each subcomponent x_i at position i of x that is of type principal there is a corresponding subcomponent y_i at the same position i of y that is also of type principal and at least one of the following also holds:

- If x_i is a trusted third party (TTP) then yi is also a trusted third party (TTP).
- If x_i is the generator of x then yi is the generator of y.

- If x_i is not the generator of component x then y_i is not the generator of y.
- If x_i is the intended recipient of x then y_i is the intended recipient of y.
- If x_i is not the intended recipient of x then y_i is not the intended recipient of y.

(A7) $Pvte(x,y) \leftrightarrow$
$\forall G_i, R_i \mid C(x,G_i,i), C(y,R_i,i) : (G_i \in ENT(P) \rightarrow R_i \in ENT(P)) \wedge$
$((G_i = TTP \wedge R_i = TTP) \vee (\exists Sl, Sq \in P : (Gen(G_i,x,Sl) \wedge$
$Gen(R_i,y,Sq)) \vee (\neg Gen(G_i,x,Sl) \wedge \neg Gen(R_i,y,Sq)) \vee$
$(R\text{int}(G_i,x,Sl) \wedge R\text{int}(R_i,y,Sq)) \vee (\neg R\text{int}(G_i,x,Sl) \wedge$
$\neg R\text{int}(R_i,y,Sq))))$

Application of axiom *(A7)* to the symmetric transformations in (4) and (5) reveals that these pairs of transformations are also principal value type equivalent:

$$Pvte(\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax)) \quad (6)$$

$$Pvte(H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP})) \quad (7)$$

The following two expressions can be derived from E3(P) and the above considered assumptions:

$$\begin{aligned} &\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax) \in CT(P) : \\ &C(m(S_1), \{N_A\}H(\{A\}datax) \wedge \\ &Gen(A, \{N_A\}H(\{A\}datax), S_1) \wedge \\ &R\text{int}(TTP, \{N_A\}H(\{A\}datax), S_1) \wedge \\ &C(m(S_2), \{N_{TTP}\}H(\{A\}datax) \wedge \\ &Gen(TTP, \{N_{TTP}\}H(\{A\}datax), S_2) \wedge \\ &R\text{int}(A, \{N_{TTP}\}H(\{A\}datax), S_2) \end{aligned} \quad (8)$$

$$\begin{aligned} &H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}) \in CT(P) : \\ &C(m(S_2), H(\{N_A\}H(\{A\}datax), N_A)) \wedge \\ &Gen(TTP, H(\{N_A\}H(\{A\}datax), N_A), S_2) \wedge \\ &R\text{int}(A, H(\{N_A\}H(\{A\}datax), N_A), S_2) \wedge \\ &C(m(S_3), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP})) \wedge \\ &Gen(A, H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}), S_3) \wedge \\ &R\text{int}(TTP, H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}), S_3) \end{aligned} \quad (9)$$

Axiom *(A16c)* states that two cryptographic transformation *c1*, c2 are travelling in opposite direction $\uparrow\downarrow$*(c1,c2)TOD3*, when a *TTP* is involved in the message exchange and creates one of the cryptographic transformation *c2. Further, the* intended recipient of *c2* is different from the generator of *c1.*

(A16c) $\uparrow\downarrow (c1,c2)_{TOD3} \leftrightarrow$
$\exists c1, c2 \in CT(P) : C(m(Sq), c1) \wedge Gen(G, c1, Sq) \wedge$
$R\text{int}(TTP, c1, Sq) \wedge C(m(Sr), c2) \wedge Gen(TTP, c2, Sr) \wedge$
$R\text{int}(R, c2, Sr) \wedge G \neq R$

Combining (8) and axiom *(A16c)* results in:

$$\uparrow\downarrow (\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax))_{TOD3} \quad (10)$$

Combining (9) and axiom *(A16c)* results in:

$$\uparrow\downarrow (H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}))_{TOD3} \quad (11)$$

Detection rule *(R2.3)* states that: If two principals exchange symmetric parent cryptographic transformations c1, c2, which are principal value type equivalent pairs and are travelling in opposite directions and a TTP is involved in the exchange and creates one cryptographic transformation c2, where the intended recipient of c2 is different to the generator of c1, then a parallel session attack can be can be mounted on En(P).

(R2.3) $\exists c1, c2 \in CT(En(P)) : \Pi(c1) \wedge \Pi(c2) \wedge Symmetric(c1,c2) \wedge$
$Pvte(c1,c2) \wedge \uparrow\downarrow (c1,c2)_{TOD3}$
$\rightarrow$ Parallel SessionAttack $(En(P))$

Combining (5), (6), and (10), the prerequisite of rule *(R2.3)* can be derived:

$$\begin{aligned} &\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax) \in CT(E_3(P)) : \\ &\Pi(\{N_A\}H(\{A\}datax)) \wedge \Pi(\{N_{TTP}\}H(\{A\}datax)) \wedge \\ &Symmetric(\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax)) \wedge \\ &Pvte(\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax)) \wedge \\ &\uparrow\downarrow (\{N_A\}H(\{A\}datax), \{N_{TTP}\}H(\{A\}datax))_{TOD3} \end{aligned} \quad (12)$$

In addition, combining (4), (7), and (11), the prerequisite of rule *(R2.3)* can also be derived:

$$\begin{aligned} &H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}) \in CT(E_3(P)) : \\ &\Pi(H(\{N_A\}H(\{A\}datax), N_A)) \wedge \Pi(H(\{N_{TTP}\}H(\{A\}datax), N_{TTP})) \wedge \\ &Symmetric(H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP})) \wedge \\ &Pvte(H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP})) \wedge \\ &\uparrow\downarrow (H(\{N_A\}H(\{A\}datax), N_A), H(\{N_{TTP}\}H(\{A\}datax), N_{TTP}))_{TOD3} \end{aligned} \quad (13)$$

Hence, (12) and (13) indicate that *parallel session attacks (P) can be mounted* on E_3(P).

In summary application of the the Attack Detection Logic reveals weaknesses in the design of LKY protocol that are exploitable by a replay attack (R) and a parallel session attacks (P).

3) Reasons for detected design weaknesses

In addition to detecting the presence of the design weaknesses, the logic also identifies the reasons for the failure.

As shown above, the LKY scheme is vulnerable to a replay attack as it violates freshness rule (R1.2). This violation is due to the fact that the cryptographic expression in step 2 {Nttp}H({A}datax) (as revealed in equation (2)) does not contain any component which receiver A recognizes as being fresh. The impact of this replay attack is that an attacker, without knowing any secret of a remote user, can masquerade as a legitimate remote user and can obtain the valid authentication message from any normal session between the remote user and the system TTP.

Additionally, the LKY scheme violates the symmetry rule (R2.3), therefore it is vulnerable to parallel session attacks. This violation is due to the symmetrical structure of (i) the pair of cryptographic expressions {Na}H({A}datax) and {Nttp}H({A}datax) (as shown in equation (5)) and (ii) the pair of hashed expressions H({Na}H({A}datax),Na) and H({Nttp}H({A}datax), Nttp) (as shown in equation (4)). The impact of these parallel session attacks is that an intruder can masquerade as a legitimate remote user and fool the server into accepting a login request from a user who is not registered with the system.

4) Re-designing protocol

As shown in the previous sections the LKY scheme cannot be considered secure. We now present an amended version of the LKY scheme to overcome the described weaknesses.

In order to prevent the potential replay, attack the cryptographic messages transmitted in the scheme needs to be freshness protected. Hence, to prevent triggering detection rule (R1.1) the cryptographic message {Nttp}H({A}datax) in step 2 should include a component which the recipient recognises as fresh. This can be achieved by including nonce Na, previously generated by A in step 1, in the second message of step 2. Thus, A can establish whether the received cryptographic expression belongs to the current protocol run. Consequently, any attempt by an intruder to replay the second message of step 2 will fail, as A can identify the replay through the incorrect value of Na.

In order to prevent the potential parallel session attacks, the cryptographic transformations transmitted need to be asymmetric. Hence, to prevent triggering detection rule (R2.3) the symmetrical structure of the pair of hashed expressions in steps 2 and 3 and the symmetrical structure of the pair of cryptographic expressions in steps 1 and 2 of the scheme needs to be broken. This can be achieved by adding nonce N_A and identity A to these components as shown in Fig. 1. Further, re-application of the proposed detection logic to the amended scheme demonstrates that none of the rules are triggered and hence it is deemed to be free of design weaknesses exploitable by replay or parallel session attacks.

$$E_3(P)\begin{cases} S_1 : A \rightarrow TTP : A, \{N_A\}H(\{A\}datax \\ S_2 : TTP \rightarrow A : H(\{N_A\}H(\{A\}datax), N_A), \{N_{TTP}, N_A\}H(\{A\}datax) \\ S_3 : A \rightarrow B : H(\{N_{TTP}, N_A\}H(\{A\}datax), N_{TTP}, A) \end{cases}$$

Fig. 1. Amended version proposed for LKY scheme.

III. Automation of Attack Detection Logic Theory

While logics for verifications purposes ("conventional verification logics") are powerful techniques for establishing that a design meets its specifications, the manual application of verification logics often requires in-depth expertise. A logic-based verification, as shown in the previous section, will typically include initially specifying the initial assumptions, protocol steps, and protocol goals in the language of the logic. The final and most complex step concerns the application of the logical rules and axioms, using deductive reasoning, to establish the beliefs, knowledge, and possessions of the protocol's principals. The verification process is thus complex, tedious and prone to error. This is a serious issue, as a single mistake during any stage of the verification process can render the result of the verification useless.

Automated techniques that carry out the deductive reasoning by automatically applying the logic axioms and rules offer a range of benefits including reducing the potential for human errors during verification, while simultaneously removing the need of in-depth knowledge of the employed verification logic. Also, the effort involved in protocol verification can be considerably reduced, since familiarity with the axioms and rules is no longer required. The time taken to perform the verification is greatly reduced as software can automatically verify a system in minutes while a similar manual proof often requires hours, days or even weeks.

This section outlines a prototype implementation of the Attack Detection Logic theory [21], which was integrated into an existing logic-based verification tool CDVT [22]. The CDVT tool uses a process of deductive reasoning based on Layered Proving Tree theoretical concept [17] to produce the verification results. The resulting automated system, as shown in Fig. 2, enables both attack detection analysis and conventional logic-based protocol verification from a single protocol specification.

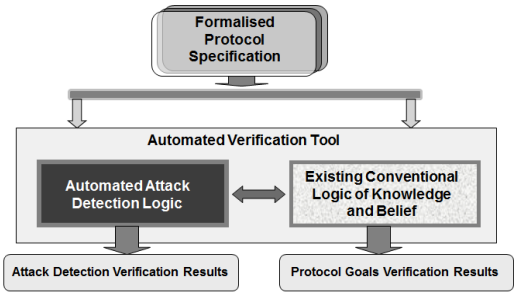

Fig. 2. Automated system overview.

A. Role of Proposed Automated Technique in Formal Verification

The scope of the new proposed automated technique is to expand the capabilities of logic-based formal verification techniques by adding attack detection to their traditional role of proving that protocols meet their security goals. This attack detection is provided by automating the detection logic that can cooperate with conventional verification logics, as outlined in Fig. 3.

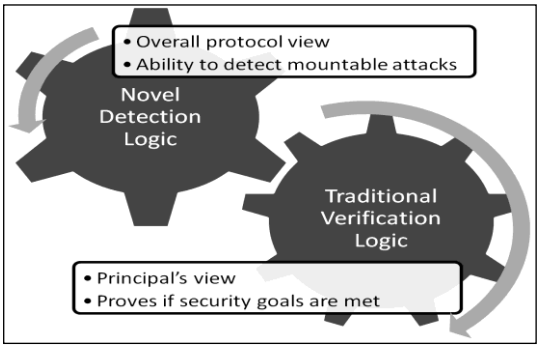

Fig. 3. Cooperation of detection and verification logics.

Together the detection logic and the conventional logic have the capability to achieve both objectives of formal security protocol verification (i.e. proving that the verified protocol meets its security goals and demonstrating the absence of mountable attacks against the protocol).

B. Structure of Automated System

The upgraded automated system, as shown in Fig. 4, comprises a number of existing modules, upgraded and new modules:

- *Existing modules* incorporate: LPT Proving Engine [17], Conventional Logic of Knowledge and Belief with adjusted postulates and termination rules [16]-[22].
- *Upgraded modules* include Unified Protocol Specification Language, Formal Specification Translator consisting of (i) Unified Grammar for Logics Specification Language; (ii) syntax validation of Unified Protocol Specification Language.
- *New module* on the implementation of the Attack Detection Logic.

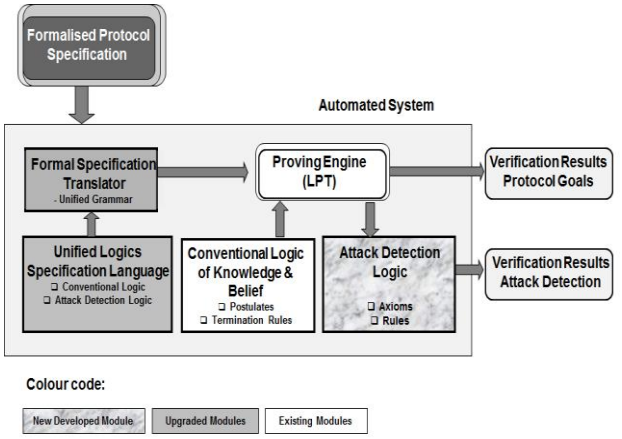

Fig. 4. Automated system modules structure.

CDVT Logic-based Verification Tool. The CDVT Verification Engine [22] is an automated system that implements a Conventional Modal Logic of Knowledge and Belief using Layered Proving Trees. The implemented logic can analyse the evolution of both knowledge and belief during a protocol execution and is therefore useful in addressing issues of both security and trust. The CDVT Verification Engine incorporates a specification language for formal protocol specifications. The engine uses a parser to read in the protocol specification from a text file, which is then processed by the LPT verification engine and the verification results are output.

Formalised Protocol Specification. The Formalised Protocol Specification introduces an efficient way for describing a security protocol to be verified within the automated system. The formalised protocol is written using English-like language (i.e. Unified Protocol Specification Language), to aid understanding and thus avoiding the complexity of the mathematical formulas as shown in previous section.

Unified Protocol Specification Language. The Unified Protocol Specification Language is an updated version of the Protocol Specification Language for Conventional Logic of Knowledge and Belief [22]. This unified language enables the formal specification of protocols that can be processed by both the Attack Detection Logic and the Logic of Knowledge and Belief. A protocol is specified declaratively in an individual text file (.txt), following well-defined syntax statements. The formalisation specification of a protocol incorporates three parts: Assumptions, Steps, and Goals. General form of specifications is:

Label : *Statement* ;

Label is An, Sn, Gn for Assumptions, Steps, Goals respectively, 'n' is positive integer (e.g. protocol assumptions defined with label A1, A2, etc., protocol steps defined with S1, S2, etc.). Moreover, each and every line must end with a semicolon (';') and comments are introduced following a double forward slash '//', i.e., the C++ style comments.

Statements are defined according to the rules presented in Table I, where elements follow the regular expressions given in Table II. "Data" is either an atomic unit or a composite data as defined in Table III. "i" indicates the indexed discrete time (-1: indicates previous protocol runs; 0: indicates the beginning of the current protocol run; 1 – n: indicates the time at step 1 – n) and "Statement" represents an arbitrary statement. "Operator" can be any of: "send", "receive" or "possess", while "Trans_Operator" are the transmission operators and can be one of the following: "send to" or "receive from". The purpose of the transmission operators is to construct a specific type of statement that expresses reception from or emission to a principal.

For example, the following step of a protocol: *S1: A -> B: A,{Na}Kab*, can formalised using the Unified Protocol Specification Language as:

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

TABLE I: STATEMENT CONSTRUCTION

Principal Operator at[i] Data
Principal Trans_Operator Principal at[i] Data
Principal know at[i] Statement
Principal believe at[i] Statement
Principal know at[i] NOT (Statement)
Principal believe at[i] NOT (Statement)
(Statement)
NOT(Statement)
(Statement AND Statement)
(Statement IMPLY Statement)

TABLE II: ATOMIC UNITS TEXTUAL GRAMMAR

Textual Grammar	**Regular Expression**
Principal	[AB-EIJLMOQRSU-Z][A-Za-z_0-9_]*
Trusted Principal	TTP[A-Za-z0-9_]*
Sym. Key	K[a-z][a-zA-Z0-9_]*
Public Key	K[a-z][A-Za-z0-9_]*Pub
Private Key	K[a-z][A-Za-z0-9_]*Priv
Nonce	N[a-z][A-Za-z0-9_]*
Timestamp	TS[a-z][A-Za-z0-9_]*
Function	F[A-Za-z0-9_]*
Hash	H[A-Za-z0-9_]*
Binary Data	[a-z][A-Za-z0-9_]*

Formal Specification Translator. The Formal Specification Translator uses the Formalised Protocol Specification to automate the creation of the data input sets required by Layered Proving Trees Proving Engine. The translator also verifies the syntax of the protocol specification. The Layered Proving Trees Engine (LPT) produces the results for the Conventional Logic of Knowledge and Belief (i.e. prove the correctness of protocol goals). It also provides the data sets required by the rules of the Attack Detection Logic module,

which verifies if any of the rules of the logic are violated and outputs the attack detection verification results accordingly.

TABLE III: COMPOSITE DATA CONSTRUCTION

Composite Data	Textual Representation
Concatenation	Data,Data
Group Element	(Data)
Symmetric Encryption	{Data}Data
Public Key Encryption	{Data}KPub
Private Key Encryption	{Data}KPriv
Function of Data	F(Data)
Hash of Data	H(Data)
Key Material of Data	KMaterial(Data)

The Formal Specification Translator requires defining a grammar for the protocol specification language to allow protocol formalisation using the textual statements. The grammar for the Conventional Logic of Knowledge and Belief was updated with the additional requirements of the Attack Detection Logic to create a Unified Grammar for both logics. Additionally, the new grammar required that also the validation process of the syntax and semantics of the formal specification to be updated. The updated parser automates the creation of the statements used as input by the LPT engine, and also enables the validation of the syntax and semantics of the formal specification of the protocol. The protocol formalisation translator has three-phases: (1) The lexical phase allows the creation of lexical units by recognising lexical patterns from the protocol formalisation. Lexical patterns are defined using regular expressions; (2) The parsing phase that allows grouping lexical units into syntactic units and construction of a parse tree representation of the protocol; (3) The semantic analysis phase allows the analysis of the parse tree for context-sensitive information and construction of an annotated parse tree. A symbol table is used to store variable and objects used to perform context-sensitive checking.

Implementation of Attack Detection Logic. The Attack Detection Logic was implemented in C++. The function of the Attack Detection Logic module is to take a data set (from the LPT), which incorporates the formalised protocol assumptions and steps, and apply the axioms in order to derive the perquisites of the set of the attack detection rules. If all the prerequisites of a rule can be established, then a weakness message leading to a replay or parallel session attack is output to the results.

IV. EVALUATION STUDY ON PROPOSED TECHNIQUE

This section presents empirical verification results of a range of security protocols, obtained from the prototype automated implementation of the Attack Detection Logic. The prototype was executed on a PC with 2GHz Intel Core™2 Duo processor and 2GB RAM, running Windows 7. The analysis results are summarized in Table IV, where the second column enumerates previously published replay (R) or parallel session (P) attacks on the analyzed protocols. The third column indicates the rules of the detection logic that reveal the attack(s), while the last column presents the verification time in milliseconds. The protocols evaluated include those with known weaknesses and their published amended versions. The technique can be considered effective if:

- Protocols with known weaknesses trigger some of the attack detection rules.
- Protocols without weaknesses do not trigger any attack detection rule.

This study shows that the Automated Attack Detection Logic is able to detect all previously published weaknesses exploitable by replay and parallel session attacks in the chosen set of security protocol with known weaknesses. Also the protocols (e.g. amended versions) for which no replay or parallel session attacks are known indicate that none of the detection rules are established, demonstrating that the detection logic does not produce false positives. Further, the measured execution times obtained for the verification of the protocols highlights the efficiency of logic-based approach, where short verification times were achieved. Another benefit of the logic-based approach is the modest memory space requirements to model these protocols and execute the verifications - the empirical study was carried out on a computing platform of 2GB of RAM.

We provide a sample verification using our proposed automated technique of the smart-card authentication protocol [9], which was manually analysed in section 2, in the Appendix.

TABLE IV: EMPIRICAL RESULTS OF ATTACK DETECTION LOGIC

Analyzed Protocol	Published Attacks	Triggered Logic Rule	Time * ms
NS PK,1978	1981[23]; 1995[4]	R1.1(R); R1.2(R); R4.7(P)	427
Lowe's fix NS PK,1995	1981[4]	R1.1(R); R1.2(R)	301
AS RPC,1989	1990[3]; 1996[4]	R1.2(R); R4.3(P)	378
BAN mod AS RPC,1990	1996[4]	R4.3(P)	411
BAN conc.AS RPC,1990	1996[4]	R4.1(P)	440
Lowe AS RPC, 1996[4]	No attack	None	503
BANYahalom,1990	1994[6]	R2.4(P)	839
Paul.Yahalom, 2001[24]	No attack	None	823
SPLICE/AS,1991	1995[25]; 1995[25]	R1.3(R); R3.4(P)	910
HC SPLICE/AS,1995	1995[25]	R1.3(R)	1051
Kao-Chow,1995	1995[26]; 2008[22]	R1.3(R); R2.1(P)	1250
W-M Frog,1994	1998[20]	R2.3(P)	103
SSH PK,1996	1997[27]	R3.3(P)	598
Abadi SSH PK, 1997[27]	No attack	None	645
PKM. IEEE 802.16, 2004	2006[28];	R1.1(R); R1.2(R)	410
PKM_v2 IEEE802.16, 2005	2006[28]	R4.1(P)	489
Lowe W-M Frog,1997	2008[5]	R1.2(R); R2.3(P); R4.3(P)	786
CBKM IC,2008	2008[29]	R1.1(R); R1.2(R);	252
DZC CBKM IC,2008[29]	No attack	None	336
KJKW IP, 2009	2013[30]	R2.2(P); R4.1(P)	201
LMLM_KJKW IP, 2012[30]	No attack	None	279
LKY Auth., 2005	2013[31]; 2007[10]	R1.2(R); R2.3(P)	993
NKPW mod. LKY, 2007	2013[31]	R1.2(R); R2.3(P)	593
JDC mod. LKY, 2013, [31]	No attack	None	602
MSCP, 2009	2018[15]	R1.1(R); R4.7(P)	492
JLCGH MSCP, 2018 [15]	No attack	None	712

* Includes combined Attack Detection and Conventional Logics verification times.

V. Conclusion

This paper concerned the formal verification of security protocols using logic-based techniques. The research objective of this work was the development of an automated logic-based technique which besides establishing the correctness of security goals of protocols is also able to demonstrate the absence of mountable attacks against the protocols. Our proposed automated technique can be used at the design stage of a security protocol to establish the presence of such weaknesses. If any weaknesses are revealed, the technique also identifies the reasons for these design weaknesses. This information can then be used to eradicate the design weaknesses. In this paper, the ability of the Attack

Detection Logic to detect weaknesses exploitable by replay and parallel session attacks was demonstrated by applying it to an authentication protocol with known weaknesses. Further, the results of an evaluation study on the effectiveness of our proposed automated technique on a range of selected protocols revealed that: (i) for all the protocols evaluated those with known replay or parallel session attacks trigger at least one of the logic detection rules, (ii) detection of all design weaknesses exploitable by the published replay and parallel session attacks, (iii) none of the detection rules were triggered for protocols that are known to be secure against replay or parallel session attacks, (iv) it establishes the efficiency of the verification technique, in terms of memory requirements (study was carried out on a computing platform of 2GB of RAM) and execution times (milliseconds) required for protocol verification.

Appendix: Formal Verification of LKY Scheme

A. Formalisation of the Authentication Scheme of Lee, Kim and Yoo (LKY) [9]

//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);

//LKY scheme steps

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);

//LKY scheme goals

G1: A know at [2] TTP send at [2] H({Na}H({A}datax),Na);
G2: A know at [2] NOT(Zero send at [0] H({Na}H({A}datax), Na));
G3: A know at [2] TTP send at[2] {Nttp}H({A}datax);
G4: A know at [2] NOT (Zero send at [0] {Nttp}H({A}datax));
G5: TTP know at [3] A send at [3] H({Nttp}H({A}datax),Nttp);
G6: TTP know at[3] NOT(Zero send at [0] H({Nttp}H({A}datax),Nttp));

B. Verification Results of the of the LKY Scheme

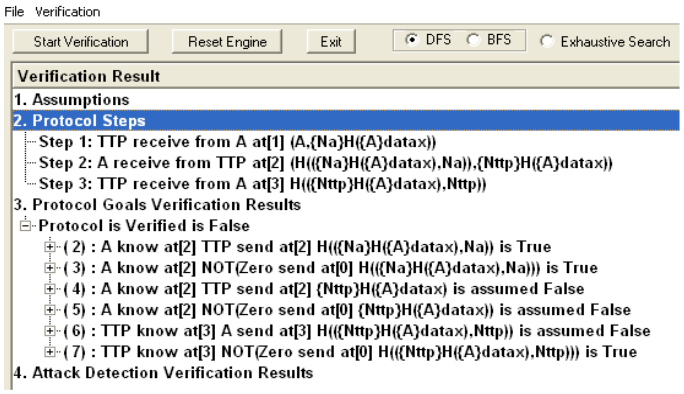

Fig. 5. Security goals verification results of LKY scheme.

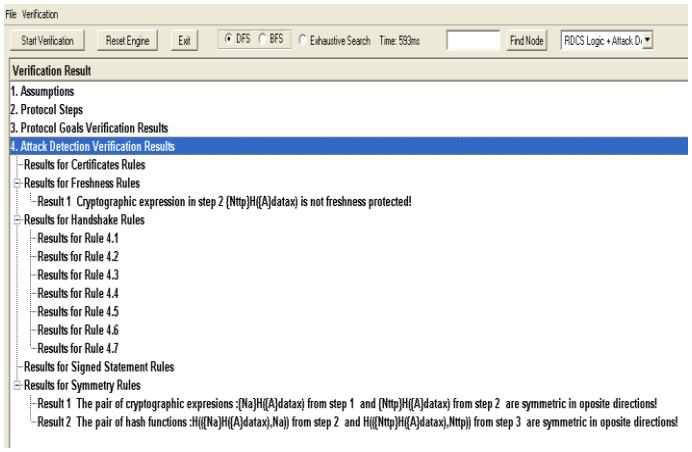

Fig. 6. Attack detection results of LKY scheme.

References

[1] A. Jurcut, T. Coffey, and R. Dojen. "Design guidelines for security protocols to prevent replay & parallel session attacks," *Computer and Security*, vol. 45, pp. 255-273, 2014.

[2] T. Coffey, R. Dojen, and T. Flanagan, "Formal verification: An imperative step in the design of security protocols," *Computer Networks Journal Elsevier Science*, vol. 43 pp. 601-618, December, 2003.

[3] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication ACM transactions on computer systems," *TOCS*, vol. 8, no. 1, pp 18-36, February 1990.

[4] G. Lowe, "Some new attacks upon security protocols," in *Proc. of the Computer Security Foundations Workshop VIII*, pp. 1996, 162-169.

[5] A. Jurcut, T. Coffey, R. Dojen, and R. Gyorodi, "Analysis of a key-establishment security protocol," *Journal of Computer Science and Control Systems*, pp. 42-47, May 2008.

[6] P. Syverson, "A taxonomy of replay attacks," in *Proc. of the Computer Security Foundations Workshop (CSFW97)*, 1994, pp. 187-191.

[7] A. Jurcut, T. Coffey, R. Dojen, and R. Gyorodi, "Security protocol design: A case study using key distribution protocols," *Journal of Computer Science and Control Systems*, vol. 2, no. 2, pp. 16-21, 2009.

[8] V. Pasca, A. Jurcut, R. Dojen, and T. Coffey, "Determining a Parallel session attack on a key distribution protocol using a model checker," in *ACM Proc. of the 6th International Conf. on Advances in Mobile Computing and Multimedia (MoMM '08),* Linz, Austria, 2008, pp. 150-155.

[9] S. Lee, H. Kim, and K. Yoo, "Efficient nonce-based remote user authentication scheme using smart cards," *Appl. Math., Comput.*, vol. 167, no. 1, pp. 355-361, August 2005.

[10] J. Nam, S. Kim, S. Park, and D. Won, "Security analysis of a nonce-based user authentication scheme using smart cards," *IEICE Transactions Fundamentals*, vol. E90-A, no. 1, pp. 299-302, January 2007.

[11] A. Jurcut, T. Coffey, and R. Dojen, "On the prevention and detection of replay attacks using a logic-based verification tool," in *Proc. of International Conf. on Computer Networks, Computer Networks, Series: Communications in Computer and Information Science*, pp. 128-137, vol. 431, June 2014.

[12] A. Jurcut, T. Coffey, and R. Dojen, "Symmetry in security protocol cryptographic messages – A serious weakness exploitable by parallel session attacks," in *Proc. of 7th IEEE International Conf. on Availability, Reliability and Security (ARES'12)*, August 2012.

[13] I. Androulidakis, "Sms security issues," *Mobile Phone Security and Forensics*, pp. 71-86, 2016.

[14] S. Wu and C. Tan, "High security communication protocol for sms," in *Proc. of International Conf. on Multimedia Information Networking and Security*, 2009, pp. 53-56.

[15] A. Jurcut, M. Liyanage, J. Chen, C. Gyorodi, and J. He, "On the security verification of a short message service protocol," in *Proc. of IEEE Wireless Communications and Networking Conf. (WCNC)*, April, 2018.

[16] T. Coffey and P. Saidha, "Logic for verifying public-key cryptographic protocols," *IEEE Proceedings-Computers and Digital Techniques*, vol. 144, no. 1, pp. 28-32, 1997.

[17] R. Dojen and T. Coffey, "Layered proving trees: A novel approach to the automation of logic-based security protocol verification," in *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, no. 3, pp. 287-311, 2005.

[18] A. Datta, A. Derek, J. Mitchell, and A. Roy, "Protocol composition logic (PCL)," *Electron. Notes in Theoretical Computer. Science*, vol. 172, pp. 311-358, 2007.

[19] D. Basin, S. Mödersheim, and L. Vigano, "OFMC: A symbolic model checker for security protocols," *International Journal of Information Security*, vol. 4, no. 3, pp. 181-208, June 2005.

[20] G. Lowe, "Casper: A compiler for the analysis of security protocols," *Journal of Computer Security*, vol. 6, pp. 53-84, January 1998.

[21] A. Jurcut, T. Coffey, and R. Dojen, "A novel security protocol attack detection logic with unique fault discovery capability for freshness attacks and interleaving session attacks," *IEEE Transactions on Dependable and Secure Computing*, July 2017.

[22] R. Dojen, I. Lasc, and T. Coffey, "Establishing and fixing a freshness flaw in a key-distribution and authentication protocol," in *Proc. of IEEE International Conf. on Intelligent Computer Communication and Processing*, 2008, pp. 185-192.

[23] D. Denning and G. Sacco, "Timestamps in key distributed protocols," *Communication of the ACM*, vol. 24, no. 8, pp. 533-535, 1981.

[24] L. Paulson, "Relations between secrets: Two formal analyses of the yahalom protocol," *Journal of Computer Security*, vol. 9, no. 3, pp. 197-216, 2001.

[25] J. Clark and J. Jacob, "On the security of recent protocols," *Information Processing Letters*, vol. 56, no. 3, pp. 151-155, 1995.

[26] I. Kao and R. Chow, "An efficient and secure authentication protocol using uncertified keys," *Operating Systems Review*, vol. 29, no. 3, pp. 14-21, 1995.

[27] M. Abadi, "Explicit communication revisited: Two new attacks on authentication protocols," *IEEE Transactions on Software Engineering*, vol. 23, no. 3, pp. 185-186, 1997.

[28] S. Xu and C. Huang, "Attacks on PKM protocols of IEEE 802.16 and its later versions," Computer Science and Engineering Department, University of South Carolina, Columbia, 2006.

[29] R. Dojen, F. Zhang, and T. Coffey, "On the formal verification of a cluster based key management protocol for wireless sensor networks," in *Proc. of 27th IEEE International Performance Computing and Communications Conference (IPCCC08), Workshop of Information and Data Assurance (WIDA08)*, Austin, Texas, USA, 2008, pp. 499-506.

[30] C. Lv, M. Ma, H. Li, and J. Ma, "A security enhanced authentication and key distribution protocol for wireless networks," *Security and Communication Networks*, vol. 5, no. 4, pp. 343-352, 2012.

[31] A. Jurcut, T. Coffey, and R. Dojen, "Establishing and fixing security protocols weaknesses using a logic-based verification tool," *Journal of Comunication*, vol. 8, no. 11, pp. 795-806, 2013.

Anca D. Jurcut received a bachelor of mathematics and computer science from West University of Timisoara, Romania (2007) and a Ph.D from University of Limerick, Ireland (2013). From 2008 to 2013, she was a research assistant with the Data Communication Security Laboratory at University of Limerick, and from 2013 to 2015, she was working as a postdoctoral researcher in the Department of Electronic and Computer Engineering at the University of Limerick and as a software engineer at IBM, Ireland. Since 2015, she has been an assistant professor with the School of Computer Science, University College Dublin, Ireland. Her research interests focuses on network and data security, security for internet of things (IoT), security protocols, formal verification techniques and applications of blockchain technologies in cybersecurity.