Research Article

Logic-Based Security Architecture for Systems Providing Multihop Communication

Iman Almomani,1 Eman Al-Banna,1 and Mousa Al-Akhras2

1 Computer Science Department, King Abdullah II School for Information Technology, The University of Jordan, Amman 11942, Jordan
2 Computer Information Systems Department, King Abdullah II School for Information Technology, The University of Jordan, Amman 11942, Jordan

Correspondence should be addressed to Iman Almomani; i.moman@ju.edu.jo

Received 16 June 2012; Revised 1 September 2012; Accepted 2 September 2012

Academic Editor: Leonardo B. Oliveira

Copyright © 2013 Iman Almomani et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Security is a basic element of distributed systems such as ad hoc and sensor communication networks. Several standards define security requirements and enforcers, such as ITU-T Recommendations X.800 and X.805. It is essential to specify and analyze protocols to know which security requirements they achieve. This paper presents a logic-based security architecture (LBSA). LBSA is a systematic way to test if a protocol is secure by checking what security requirements are achieved. Different rules, actions, and sets which fit into the proposed LBSA are included, new ones are also added to complete the architecture. The key advantage of LBSA is that it enables a security protocol to prove its correctness mathematically. Mathematical proofs provided by LBSA cover more cases that usually cannot be covered exhaustively by simulation tools. This paper also specifies and analyzes several security enforcers and protocols and mathematically proves which security requirements they achieve. Mapping between security requirements and inference rules/actions is also provided to facilitate the use of LBSA. Some enforcers are analyzed using LBSA to demonstrate how they achieve security requirements. Finally, we take Ariadne protocol as a case study and show how to use the proposed LBSA architecture to prove that this protocol is secure.

1. Introduction

When two entities communicate to obtain a certain service(s), they must ensure secure end-to-end communication. Systems do not provide efficient services without applying proper security mechanism due to the existence of different types of attackers. Security can be defined through a set of requirements that must be achieved by the communicating parties to communicate securely and protect services from attackers.

Because of the importance of security for end-to-end communication, many secure protocols have been proposed as will be discussed later in the paper. Some of these protocols had taken prevention measures to stop attackers while others had taken the detection approaches. A way to analyze these protocols is required to check if these protocols are secure as their designers claim and to know which security requirements these protocols achieve. In this paper, we propose a logic-based security architecture (LBSA) which is an easy, fast, and reliable way to specify and analyze secure protocols.

Some security requirements, as defined by several standards such as ITU-T Recommendations X.800 and X.805 [1, 2], must be achieved to declare that a protocol is secure. Using LBSA, a protocol can be tested to check which security requirements it achieves. Several efforts have been done to utilize logic in such test [3–8], but logic was used to specify and analyze specific protocols by defining global and local sets, actions, and inference rules. In this paper security architecture is taken as a whole, different actions, sets, and rules which fit into the architecture have been included, that means specifying and analyzing different enforcers and determining the requirements they satisfy. Also, we had labeled the actions, sets and rules for easy access and we added our own to complete the architecture. Additionally, we mapped different security requirements into inference rules/actions. This mapping shows the inference rules/actions
which are used during the protocol analysis to achieve security requirement(s).

Utilizing LBSA, protocols can be tested to check if they achieve the security requirements specified by their designers. This checking can be performed by analyzing the protocol and applying appropriate actions and rules. If the rules are applied successfully we conclude that their claim is true, otherwise it is false.

The rest of this paper is organized as follows: Section 2 illustrates the security architecture. Section 3 presents related work. Section 4 presents the proposed Logic-Based Security Architecture (LBSA) for systems providing multihop communication. Section 5 provides possible mapping between security requirements and inference rules/actions. In Section 6, the specification and analysis of message authentication code (MAC) and digital certificate based on LBSA are discussed. Section 7 illustrates how to specify and analyze protocols using LBSA by taking Ariadne routing protocol as a case study. Finally, Section 8 concludes the paper and presents avenues for future work.

2. Security Architecture

Security architecture as defined by ITU-T Recommendations X.800 [1] and X.805 [2] defines a set of security requirements that must be achieved. Figure 1 illustrates this architecture.

Security architecture as shown in Figure 1 includes a set of security requirements that are used to protect systems from different types of attackers. To achieve the objectives of these requirements, a set of security enforcers must be applied.

The requirements and their definitions are illustrated as the following.

(i) **Authentication**: Prove the identity of the communicating parties.

(ii) **Authorization**: control the access to the system's resources. Give different entities different privileges according to their roles.

(iii) **Confidentiality**: ensure the secrecy of data. Secret data must be read only by intended recipients.

(iv) **Integrity**: protect the received data from any kind of modifications during transmission from the sender to the receiver.

(v) **Nonrepudiation**: prevent the users from denying the sending of messages or initiating events that they had performed.

(vi) **Privacy**: protect the identity and/or the location of the node and sometimes the routing protocol being used.

(vii) **Availability**: ensure that no one prevents authorized users from getting access to the system available services.

These requirements must be achieved to guarantee secure communication and to prevent and detect different attackers which can be classified into the following.

(i) **Internal or external attackers**: internal attackers are compromised users who are authorized to enter the system but they are misbehaving. This type of attack needs a detection mechanism to be discovered as they have the authority to access the system's services. External attackers, on the other hand, are unauthorized entities attempting to access the system's services; these attackers must be prevented from accessing the system's resources.

(ii) **Passive or active attackers**: passive attackers monitor the system without taking any action and usually it is a phase that precedes an active attacking. Active attacker takes actions and does modifications.

(iii) **Intentional or accidental**: intentional attack is a planned attack while accidental attacking results from system malfunctions such as software bugs.

Several mechanisms and techniques have been defined to achieve different security requirements. Table 1 gives examples of different security enforces that are used to implement the objectives of different security requirements.

Secure protocols usually use the above enforces, among others, to achieve the security requirements. There are different ways to analyze security protocols in terms of achieved security requirements such as simulation and mathematical models; this paper uses logic to do such analysis.

### Table 1: Examples of security requirements enforcers

| Security requirement | Security enforcer |
|----------------------|------------------|
| **Authentication**   | (i) Digital certificate  |
|                      | (ii) Digital signature  |
|                      | (iii) Message authentication code (MAC)  |
| **Authorization**    | Firewall  |
| **Confidentiality**  | Encryption  |
| (i) Hash function  |
| (ii) MAC  |
| (iii) Digital signature  |
| **Integrity**        | Digital signature  |
| **Nonrepudiation**   | (i) Achieved partially by encryption  |
|                      | (ii) Hide node location  |
|                      | (iii) Hide node identity  |
|                      | (iv) Hide routing protocol  |
| **Privacy**          | (i) Hide node location  |
| (ii) Hide node identity  |
| (iv) Hide routing protocol  |
| **Availability**     | (i) Intrusion detection scheme (IDS)  |
|                      | (ii) Intrusion prevention scheme (IPS)  |

3. Related Work

The importance of security in providing successful services in any distributed system raises the necessity of having formal way to analyze security protocols. Previous effort in using logic for analyzing security is Rubin logic [3, 4].

Rubin logic is an approach that specifies and analyzes nonmonotonic cryptographic protocols. It is one of the first approaches to allow reasoning about nonmonotonic protocols [3]. In nonmonotonic protocols, beliefs are changed...
Security architecture

Security requirements
- Authentication
- Authorization
- Confidentiality
- Integrity
- Nonrepudiation
- Privacy
- Availability

Security attackers
- Internal and external
- Passive and active
- Intentional and accidental

Security enforcers
- Digital certificate
- Digital signature
- Firewall
- Encryption
- Hash function
- MAC
- IDS/IPS

Figure 1: Security Architecture extracted from ITU-T Recommendations X.800 and X.805.

during protocol execution time. An example of nonmonotonicity is the belief that a key must be changed when a node becomes compromised. To achieve the protocol specification and analysis, Rubin logic defines global and local sets, actions and inference rules.

Rubin and Honeyman [3] focused on authentication protocols. They took KHAT protocol [9] as an example to discover its flaws. KHAT protocol was built to solve the problem of long running jobs in an authenticated environment where a trusted server issues tickets with limited lifetimes for services. The authors gave special attention to ensure the freshness of data using fresh nonce. The main problem they attempted to solve is that principal B cannot achieve the belief that the session key with principal A is fresh. Finally, the authors defined most of global and local sets that are used later in the literature.

Rubin [4] extended the work presented in [3] by adding one set. Rubin [4] aimed to make sure that keys are observed by their intended parties and data items are fresh, especially the public keys. Rubin [4] made link between certificates and requests which reveals weakness in Needham and Schroeder public key protocol [10].

Xu and Xie presented in a series of papers [5–8] the utilization of Rubin logic in analyzing the security for specific protocols.

In [5], Xu and Xie extended the work presented in [4] to analyze nonrepudiation in routing protocols proposed for wireless mobile ad hoc networks (MANET). This work took ARAN [11] routing protocol to test nonrepudiation.

Xu and Xie [6] use the work presented in [5] to analyze electronic commerce protocols and in [7] they have chosen Zhou-Gollmann [12] protocol which is a simple and effective nonrepudiation protocol to illustrate how an electronic commerce protocol is analyzed using the extended Rubin logic.

Two examples of Rubin logic’s applications are given by Xu and Xie in [8]. First example is the Andrew secure RPC [13] protocol using symmetric keys. The second one is X.509 [14] authentication protocol using asymmetric keys.

As can be illustrated from the above-related work, all attempts to utilize Rubin logic have either focused on a specific requirement or a specific protocol. This paper proposes a logic-based security architecture (LBSA) that presents a formal way to analyze any security requirement in any system providing multihop communication. All sets, actions, and rules presented in previous efforts are considered and generalized; new ones are added to complete the architecture. After that, we illustrate how LBSA will be used to test security requirements and issues in different security enforcers and protocols.

4. Logic-Based Security Architecture (LBSA) for Systems Providing Multihop Communication

In this section we illustrate the proposed architecture which defines a logical way for specifying and analyzing different enforcers and different protocols that achieve any security requirement as defined in [2]. Some researchers used logic as mentioned earlier in Section 3. All sets, actions, and rules defined in [3–8] which fit in our architecture have been included and labeled by the proposed LBSA.

Global sets define the specification of the protocol as a whole. Local sets are private to each principal in the specification. Actions are specified as part of the protocol (i.e., how the protocol works) while inference rules are used to reason about the beliefs during the protocol executing. Consequently, the relation between sets, actions, and rules and the result of the action may update the sets. This achieves some rules and conditions which are followed by applying inference rules which in turn update another set(s). Figure 2 shows this relation.

Using different actions, the protocol can be specified exactly as it works. While executing these actions, local and global sets will have new values which lead to applying some rules. This is the process of analyzing protocols.

The purpose of LBSA is to generalize rules and actions and not to specify them. Accordingly, these rules and actions can be customized according to the context. Sections 6 and 7 illustrate how these sets, rules, and actions can be customized according to the context. Additionally, each set, rule, and action are described in the following tables using the description column.

Table 2 defines different global and local sets. Global sets are labeled as GS, and local sets are labeled as LS, where i is
4 International Journal of Distributed Sensor Networks

Table 2: Global and local sets of LBSA.

| Set label | Set name | Description |
|-----------|----------|-------------|
| GS₁       | Principal set | All principals (i.e., nodes) participate in the protocol are in this set. $P = \{P₁, P₂, \ldots, Pₙ\}$. There is one initiator of the protocol and it could be any $Pᵢ$. |
| GS₂       | Rule set | Inference rules needed to derive new statements from existed assertions that are in this set. $R = \{R₁, R₂, \ldots, Rᵢ\}$, where $Rᵢ$ is of the form $(Cᵢ, Cᵢ, \ldots, Cᵢ) / D$ where $Cᵢ$ is a condition and $D$ is a statement. |
| GS₃       | Secret set | Each secret in the system exist in this set. During the analysis, the secret content of this set will be changed for the emergence of new secrets such as session keys. $S = \{S₁, S₂, \ldots, Sᵢ\}$. |
| GS₄       | Observers set | There are five global sets GS₁–GS₅ where the set number is 4. These sets contain the protocol information. There are five global sets GS₁–GS₅ and ten local sets LS₁–LS₁₀. These sets have initial values and these values are changed whenever actions and rules are executed. Global sets (GS₁–GS₅) were initilly defined in [3] for a specific protocol called KHAT [9]. These global sets have been generalized by LBSA and new global sets are also added. Local sets (LS₁–LS₉) were initially defined in [3–5] for specific protocols, but they are generalized and labeled by LBSA which also defines new local sets needed to complete the architecture. |
| GS₅       | FireWall set($Pᵢ$) | This set presents the firewall list that contains the access control rules that are needed to be applied to the incoming packets of $Pᵢ$ in order to allow or prevent them. These sets contain all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₁       | Possession set($Pᵢ$) | This set contains all the beliefs held by a principal. This includes the belief that the keys it holds between itself and other principals are good, beliefs about jurisdiction, beliefs about freshness, and beliefs about the possessions of other principals. BEL($Pᵢ$) = \{bel₁, bel₂, \ldots, belᵢ\}. |
| LS₂       | Belief set($Pᵢ$) | This set contains candidates to be added to the Seen set. It is used by the Update function (Table 3(b)). The Seen set contains text parts (data) of the message and a list of the associated keys needed to derive new statements from existed assertions. LS₂ is a list rather than a set because the elements are ordered. LS₂ = \{BL₁, bvr₁, bvr₂, \ldots, bvrᵢ\}. |
| LS₃       | Opaque($Pᵢ$) | This set contains all the other principals’ nonrepudiation that $Pᵢ$ can prove. LS₃ contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₄       | Seen($Pᵢ$) | This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₅       | Haskeys($Pᵢ$) | This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₆       | Behavior list($Pᵢ$) | This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₇       | Bindings set($Pᵢ$) | This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₈       | Proofs set($Pᵢ$) | This set contains all the other principals’ nonrepudiation that $Pᵢ$ can prove. LS₈ contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. POSS($Pᵢ$) = \{poss₁, poss₂, \ldots, possᵢ\}. Note: possᵢ contains two fields—the actual data and the origin of this data. |
| LS₉       | PL($Pᵢ$) | Principals list contains all the principals that must receive Messages (Msgs) from principals $Pᵢ$. |
| LS₁₀      | AbnBeh($Pᵢ$) | Abnormal behavior set contains all the principals that $Pᵢ$ detect their abnormal behaviors. |

Figure 2: Relation between sets, actions and inference rules.
Table 3: (a) Actions (ACT₁ – ACT₁₀) of LBSA, (b) actions (ACT₁₁ – ACT₂₀) of LBSA, and (c) actions (ACT₂₁ – ACT₃₀) of LBSA.

(a) Action label | Action name | Description | Condition | Result |
---|---|---|---|---|
ACT₁ | Send\((P_j, X)\) | \(P_j\) sends msg \(X\) to \(P_j\). | — | Principal \(P_j\) sends msg \(X\) to \(P_j\). |
ACT₂ | Receive\((P_j, X)\) | \(P_j\) receives msg \(X\) from \(P_j\). | — | Principal \(P_j\) receives msg \(X\) from \(P_j\). |
ACT₃ | Encrypt\((X, k)\) | If \(P_j\) possesses msg \(X\) and knows the key \(k\) then it can encrypt \(X\) using \(k\) and possess \(\{X\}_k\). Observers \((k)\). | \(X, k \in \text{POSS}(P_j), P_j \in \text{Observers}(k)\). | POSS\((P_j) = \text{POSS}(P_j) \cup \{\{X\}_k\}\). |
ACT₄ | Decrypt\((\{X\}_k, k)\) | If \(P_j\) possesses msg \(X\) which encrypted under \(k\) and knows the key \(k\) (symmetric encryption) or its inverse (Asymmetric encryption), then \(P_j\) can decrypt \(X\) and possess it. | POSS\((P_j) = \text{POSS}(P_j) \cup \{X\}\). |
ACT₅ | Generate-nonce\((N)\) | Is used to generate nonce \(N\) to link a challenge and a response. When the response is received, \(\text{LINK}(N)\) is removed from \(\text{BEL}(P_j)\). | — | POSS\((P_j) = \text{POSS}(P_j) \cup \{N\}\), \(\text{BEL}(P_j) = \text{BEL}(P_j) \cup \text{LINK}(N)\). |
ACT₆ | Generate-secret\((s)\) | Is used to generate a new secret \(s\). Note that the Observers set will be updated. | — | \(S := S \cup \{s\}\), \(\text{Observers}(s) := \text{Observers}(s) \setminus \{P_j\}\), \(\text{POSS}(P_j) := \text{POSS}(P_j) \setminus \{s \setminus P_j\}\), \(\text{BEL}(P_j) := \text{BEL}(P_j) \cup \#(s)\), where \# indicates that the value is fresh. |
ACT₇ | Concat\((x_1, x_2, \ldots, x_n)\) | Is used to construct a message \(X\) out of submessages \(x_1, x_2, \ldots, x_n\). When a message contains a set of components, this action is used to break it into its components. | \(x_1, x_2, \ldots, x_n \in \text{POSS}(P_j)\). | POSS\((P_j) = \text{POSS}(P_j) \cup \{x_1, x_2, \ldots, x_n\}\). |
ACT₈ | Split\((X)\) | This action is used to break \(X\) into its components. When \(P_j\) no longer possesses \(X\), this action will be used. If other principals trust \(P_j\), then they can believe that \(P_j\) is no longer in possession of \(X\). | \(X \in \text{POSS}(P_j)\). | POSS\((P_j) = \text{POSS}(P_j) \cup \{x_1, x_2, \ldots, x_n\}\). |
ACT₉ | Forget\((X)\) | Is used to forget message \(X\). When a message contains a set of components, this action is used to break it into its components. When \(P_j\) no longer possesses \(X\), this action will be used. If other principals trust \(P_j\), then they can believe that \(P_j\) is no longer in possession of \(X\). | \(X \in \text{POSS}(P_j)\). | \(\text{BEL}(P_j) := \text{BEL}(P_j) \setminus \{X\} \text{ if TRUST}[j, i] = 1\). |
ACT₁₀ | Check-freshness\((X)\) | Is used to verify the freshness of \(X\). When a message contains a set of components, this action is used to break it into its components. When \(P_j\) no longer possesses \(X\), this action will be used. If other principals trust \(P_j\), then they can believe that \(P_j\) is no longer in possession of \(X\). | \(X \in \text{POSS}(P_j), X\) not expired. | \(\text{BEL}(P_j) := \text{BEL}(P_j) \cup \#(X)\). |

(b) Action label | Action name | Description | Condition | Result |
---|---|---|---|---|
ACT₁₁ | Forget-secret\((s)\) | When \(P_j\) no longer knows the secret \(s\). If other principals trust \(P_j\), then they can believe that \(P_j\) no longer possesses \(s\). | \(P_j \in \text{Observers}(s), s \in \text{POSS}(P_j)\). | \(\text{Observers}(s) := \text{Observers}(s) \setminus \{P_j\}\), \(\text{POSS}(P_j) := \text{POSS}(P_j) \setminus \{s \setminus P_j\}\) if TRUST\([j, i] = 1\) then \(\text{BEL}(P_j) := \text{BEL}(P_j) \setminus \{s \in \text{POSS}(P_j)\}\). |
ACT₁₂ | Apply\((f, X)\) | Is used to apply function \(f\) to \(X\). The protocol aborts when different cracks and events that affect the protocol specification happen. | \(f, X \in \text{POSS}(P_j)\). | POSS\((P_j) := \text{POSS}(P_j) \cup \{f(X)\}\). |
ACT₁₃ | Abort | Protocol run is illegal. | — | Analysis reports failure. |
ACT₁₄ | Generate-key-pair\((k', k)\) | Is used to generate two keys \(k'\) (public) and its inverse \(k^-\) (private). | — | POSS\((P_j) = \text{POSS}(P_j) \cup \{k' \setminus P_j, k^- \setminus P_j\}\). |
| Action label | Action name | Description | Condition | Result |
|-------------|-------------|-------------|-----------|--------|
| ACT_{15} | Apply-Asymkey($X, k$) | There are two cases for applying asymmetric key operation on $X$. The first one when $X$ is in the form of $\{Y\}_k$, and $k$ and $k'$ are inverses to each other. The second case when $X$ is encrypted using $k$. | $X, k \in \text{POSS}(P_i)$, $k$ is an asymmetric key. | if $X = \{Y\}_k$ and $k$ is the inverse of $k'$, then $\text{POSS}(P_i) = \text{POSS}(P_i) \cup Y$ else $\text{POSS}(P_i) = \text{POSS}(P_i) \cup \{(X)_k\}$. |
| ACT_{16} | Bind($k, P_j$) | Before sending a key, principal $P_i$ can bind this key to other principle $P_j$ using this action. After this binding, both $P_i$'s possession and binding sets will contain $k \oplus P_j$. | $k \in \text{POSS}(P_i)$, $P_j \in \text{Observers}(k)$, $k$ is a key intended for $P_j$. | $\text{POSS}(P_i) = \text{POSS}(P_i) \cup \{(k \oplus P_j)\}$, $\text{Bindings}(P_i) = \text{Bindings}(P_i) \cup \{(k \oplus P_j)\}$. |
| ACT_{17} | Update($X$) | After sending a message this action is used to maintain the observers set. | — | Maintains the observers of $X$. |
| ACT_{18} | Broadcast($X$) | Message $X$ is broadcasted by $P_i$ to each of its neighbors. | — | $P_i$ broadcasts $X$ to each of its neighbors. |
| ACT_{19} | Check-certificate($\text{cert}_R$) | Is used to verify the freshness of the certificate $\text{cert}_R$. | $\text{cert}_R \in \text{POSS}(P_i)$, $\text{cert}_R$ is not expired | $\text{BEL}(P_i) = \text{BEL}(P_i) \cup \{\#(K_R)\}$, where $K_R$ is the public key specified in $\text{cert}_R$. |
| ACT_{20} | Check-request([IP_{R}, N_{R}]) | Is used to record new routing record. Each node receives route discovery packet it records the predecessor of that packet and the successor toward the destination. | IP_{R} \in \text{POSS}(P_i), N_{R} \in \text{POSS}(P_i), \text{and } \{\text{IP}_{R}, N_{R}\} \text{ is not already seen.} | $\text{POSS}(P_i) = \text{POSS}(P_i) \cup \{(\text{IP}_{R}, N_{R})\}$, $\text{Bindings}(P_i) = \text{Bindings}(P_i) \cup \{(\text{IP}_{R}, N_{R})\}$. |

### Table (c)

| Action label | Action name | Description | Condition | Result |
|-------------|-------------|-------------|-----------|--------|
| ACT_{21} | Comp($A, B$) | This action is used to compare two values if they are equal or not. | $A, B \in \text{POSS}(P_i)$ | If $A$ and $B$ are equal: $\{X\}^+ \in \text{Proof}(P_i)$, means message $X$ is checked and not modified. If they are not equal: $\text{POSS}(P_i) = \text{POSS}(P_i) - \{X\}$ means message $X$ is checked and it is modified. |
| ACT_{22} | Renew-key-pair($k^+, k^-$) | This action is used to renew the public and private keys. | — | $\text{POSS}(P_i) - \{k^+ \oplus P_i, k^- \oplus P_i\} \cup \{\#k^+, \#k^- \oplus P_i\}$. |
| ACT_{23} | Multicast($VP_i \in \text{PL}, X$) | It means that Principle $P_i$ sends msg $X$ to all principles in the PL. | — | Principle $P_i$ sends message $X$ to all principles in the PL. |
| ACT_{24} | Modify(Cert_{R}) | This action modifies certificate content. | — | Modify the certificate content. |
| ACT_{25} | Monitor($P_j$) | It means that principle $P_j$ monitors the abnormal behavior of other principals. | — | Principal $P_j$ detects abnormal behaviors of other principal $P_i, P_j \in \text{AbnBeh}(P_i)$. |
| ACT_{26} | Generate-Alert($P_i, P_j$) | $P_j$ generates an alert at time $t$ because it detects abnormal behavior of $P_j$. | $P_j \in A \text{ bnBeh}(P_i)$. | Alert($t, P_i, P_j$) |
| ACT_{27} | Alert-Filtering($t, P_i, P_j$) | This action is used to do the process of categorizing attack alerts produced from principals in order to distinguish false positive from actual attack. | Alert($t, P_i, P_j$) $\in \text{POSS}(P_i)$, where $P_i$ is any Principle in $P_i$, but not $P_i$ or $P_j$. | If true positive $P_i$ takes an action. |
| ACT_{28} | Access($\text{Resource}$) | This action is used to allow or deny the access of resources. | Access$\text{Info}(P_i) \in \text{FireWall}$, Condition = Allow | Allow resource access. |
| ACT_{29} | $X \leftarrow Y$ | This action is used to rename value $Y$ to $X$. | $Y \in \text{POSS}(P_i)$, $X \in \text{POSS}(P_i), \text{Use } X$ instead of $Y$. |
| ACT_{30} | Process($\text{Pra}, \text{Fun}$) | This action is used in key agreement process. | Fun $\in \text{POSS}(P_i)$, parameters received from $P_j$. | Key $\in \text{POSS}(P_i)$. |
Different inference rules are defined in Tables 4(a)–4(d). RL_i is the rule label where i is the rule number. As the rule conditions are executed correctly, the sets’ values will be changed or the protocol will be aborted. Rules defined for specific protocols in [3–6] had been extracted, generalized, and mapped to LBSA. New rules were also generated by LBSA to complete the security architecture.

Section 5 provides mapping between security requirements and inference rules/actions. It also provides mapping between some protocol services and inference rules/actions.

5. Possible Mapping between Security Requirements and Inference Rules/Actions

To ensure that the required security requirement is achieved by a security protocol, some inference rules must be applied and some actions must be performed. In this section we provide possible mappings between each security requirement and the inference rule that must be applied or the action that must be executed to achieve it. Table 5 summarizes these mappings.

Actions and rules are needed to complete the specification and the analysis processes. In other words, actions and rules add new values to the global and local sets which may result in achieving the mapped rules conditions.

Other actions and rules are needed to perform certain protocol services, including key management which deals with key and certificate generations, verification, and renewing. Certain services are related to nodes’ interaction and protocol functions. Data item refreshments such as messages, keys, certificates, and nonces are among the data that need to be kept fresh.

Table 6 summarizes the mappings between protocol services and the inference rule that must be applied or the action that must be executed to achieve it.

The following section presents how the above-presented architecture will be used to formally analyze security enforcers and security protocols.

6. Security Enforcers Specification and Analysis

Different security enforcers are used as part of different protocols to achieve security such as in [15–18]. In this section we have chosen well-known enforcers that can be provided at different network layers to illustrate how LBSA could be used to specify and analyze the security enforcers that could be used by many different security protocols. These enforcers are message authentication code (MAC) and digital certificate which also includes digital signature. The reason for using these enforcers is that they are the most used enforcers, in many protocols including routing protocols, to achieve authentication and integrity (MAC) and authentication, integrity and nonrepudiation (digital certificate) [19–25]. The following sections show how these enforcers could be verified using LBSA.

6.1. Message Authentication Code (MAC). MAC [26] achieves two security requirements, namely, authentication and integrity. In MAC the sender and the receiver of the message must agree on a shared key before initiating the communication. The sender of the message uses the shared key and the message as an input to a hash function to generate the message digest. Then, the sender sends the message and the digest to the receiver. The receiver checks the originality of the message and its correctness by applying the hash function on the received message and the shared key. After that, the receiver compares the resulted digest with the received message digest. If the two digests are equal, this will prove the authenticity of the message as the shared key is only known by sender and receiver. Moreover, it ensures that the message is not modified during the transmission since any simple modification will produce a different digest. To mathematically prove that MAC achieves both authentication and integrity using LBSA the following steps must be performed.

(i) Define the Values of the Global Sets. The first step in the protocol specification is to define the values of the global sets:

(GS1) \( P = \{Alice, Bob\} \) // Alice is assumed to be the initiator of the protocol.
(GS2) \( R = \{\text{contains the rules defined in the Tables 4(a)–4(d)}\} \)
(GS3) \( S = \{K_{AB}\} \) // \( K_{AB} \) is the shared key between Alice and Bob
(GS4) Observers \( (K_{AB}) = \{Alice, Bob\} \).

(ii) Define the Initial Values of the Local Sets. The second step is to define the initial values of the local sets to each principal. Note that some of the global and the local sets values will be changed during the protocol analysis.

Principal A (Alice):

LS1) \( \text{POSS}(A) = \{\text{Msg}, K_{AB}, (B \bowtie K_{AB}) \text{ from B}\} \)
LS2) \( \text{BEL}(A) = \{\#(K_{AB}), (B \bowtie K_{AB})\} \)
LS6) BL =

\[ \begin{align*}
\text{ACT}_1 & : \text{Concat}(\text{Msg, } K_{AB}) \\
\text{ACT}_2 & : \text{Apply}(H, \{\text{Msg, } K_{AB}\}) \text{ // Apply hash function to the concatenation} \\
\text{ACT}_3 & : \text{Concat}(\text{Msg, } H(\text{Msg, } K_{AB})) \text{ // prepare the message and digest} \\
\text{ACT}_4 & : \text{Send}(B, \{\text{Msg, } H(\text{Msg, } K_{AB})\}) \text{ // send message and digest to B}. 
\end{align*} \]

Principal B (Bob):

LS1) \( \text{POSS}(B) = \{K_{AB}, (A \bowtie K_{AB}) \text{ from A}\} \)
LS2) \( \text{BEL}(B) = \{\#(K_{AB}), (A \bowtie K_{AB})\} \)
LS6) BL =

\[ \begin{align*}
\text{ACT}_1 & : \text{Receive}(A, \{\text{Msg, } H(\text{Msg, } K_{AB})\}) \\
\text{ACT}_8 & : \text{Split}(\text{Msg, } H(\text{Msg, } K_{AB})) \\
\end{align*} \]
Table 4: (a) Rules (RL1–RL4) of LBSA, (b) rules (RL2–RL11) of LBSA, (c) rules (RL12–RL16) of LBSA, and (d) rules (RL17–RL19) of LBSA.

(a)

| Rule label | Rule | Description |
|------------|------|-------------|
| RL1        | Nonce verification rule: #\((X)\) ∈ BEL\((P)\), X from Q ∈ POSS\((P)\) |
|            | BEL\((P)\) = BEL\((P)\) ∪ \{Q believes #\((X)\)\} | Principle P can decide that principle Q believes that X is fresh if P's belief set contains #\((X)\) and X from Q is in the P's possession set. |
| RL2        | Message meaning rule: \(\{X\}_k\) from Q ∈ POSS\((P)\), \(\{P, Q\} \subseteq\) Observers\((k)\) |
|            | BEL\((P)\) = BEL\((P)\) ∪ \{X ∈ POSS\((Q)\)\} | P can believe that Q possesses X if it receives \(\{X\}_k\) from Q and both P and Q know the key k. |
| RL3        | Sub-message freshness rule: \#\((x_1)\) ∈ BEL\((P)\), X contains \(X\) contains \(x_1\), X contains \(x_2\) | P can believe that \(x_1\) is fresh if it believes that \(x_1\) is fresh and have \(x_1\) and \(x_2\) in the same formula and this formula is in P's possession set. That is, the part of the message that contains something fresh is fresh. |
| RL4        | Linkage rule (symmetric keys): \#\((k)\) ∈ BEL\((P)\), P ∈ Observers\((k)\), LINK\((Na)\) ∈ BEL\((P)\), X contains \(f(\text{Na})\), X contains \(x_1\), \(\{X\}_k\) from Q ∈ POSS\((P)\) |
|            | BEL\((P)\) = (BEL\((P)\) – LINK\((\text{Na})\)) ∪ \{#\((x_1)\)\} | This rule is applied only once for the same nonce since the LINK item will be removed after applying it successfully; P can believe that a submessage \(x_1\) of a message X is fresh when several conditions are applied. The first one is when P’s belief set contains LINK\((\text{Na})\) which means the nonce Na has not been used before. Also, the message X that contains the nonce must be encrypted using the key k and must be fresh and available to P; this is what other conditions state. |
| RL5        | Linkage rules (asymmetric keys): (a) \#\((k)\) ∈ BEL\((P)\), P ∈ Observers\((k)\), LINK\((Na)\) ∈ BEL\((P)\), X contains \(f(\text{Na})\), X contains \(x_1\), \(\{X\}_k\) from Q ∈ POSS\((P)\) |
|            | BEL\((P)\) = (BEL\((P)\) – LINK\((\text{Na})\)) ∪ \{#\((x_1)\)\} | This rule is applied only once for the same nonce since the LINK item will be removed after applying it successfully; P can believe that a submessage \(x_1\) of a message X is fresh when several conditions are applied. The first one is when P’s belief set contains LINK\((\text{Na})\) which means the nonce Na has not been used before. Also, the message X that contains the nonce must be encrypted using the public key \(k^*\) and the corresponding inverse (private) key \(k^*\) must be fresh and available to P; this is what other conditions state. |
| RL6        | Possible origins rule: X ∈ POSS\((P)\), X contains \(x_1\), R ∈ Observers\((x_1)\), R ≠ P |
|            | WHEN φ \(x_1\) from R ∈ POSS\((P)\) | When P’s possession set contains message X which contains submessage \(x_1\) and this submessage observed by more than one principal, other than P, it will be marked as coming from all principals by adding new item for each of the P’s Possession set. |

(b)

| Rule label | Rule | Description |
|------------|------|-------------|
| RL7        | Submessage origin rule: \(\{X\}_k\) ∈ POSS\((P)\), X contains \(x_1\) from Q, R ∈ Observers\((k)\), X contains \(x_1\), R ≠ P |
|            | \(x_1\) from Q ∈ POSS\((P)\), \(x_2\) from R ∈ POSS\((P)\) | A submessage \(x_1\) can be marked as coming from Q if P possesses a message X which contains \(x_1\) and \(x_1\) encrypted using an asymmetric key, and submessage \(x_1\) is marked as coming from Q. And any other principle who observes the inverse key, the submessage is marked as coming from it. |
| RL8        | Submessage origin rule for asymmetric keys: (a) \(\{X\}_k\) ∈ POSS\((P)\), X contains x from Q, X contains \(x_1\) |
|            | \(x_1\) from Q ∈ POSS\((P)\) | This rule is simplified version of the submessage origin rule. In (a) If P’s Possession set contains message X which is encrypted using P’s public key. And X contains a submessage \(x_1\) which is marked as coming from Q. Then all other submessages in X are marked as coming from Q. This is true because P is the only principle that has the corresponding private key so no one can change the content of X. In (b) since Q is the constructor of the message X, any submessage of X can be marked as coming from Q. This is true because Q is the only principle that has Q’s private key. |
| RL9        | Submessage origin rule for private keys: \(\{X\}_k\) ∈ POSS\((P)\), X contains \(x_1\) |
|            | \(x_2\) from Q ∈ POSS\((P)\) | }
| Rule label | Rule Description |
|------------|------------------|
| **Unbound key rule:** | When key $k$ is in $P$'s Possession set and this key is not bounded to any principal then all principals ($W$) observe it. That is, the key is not a secret. |
| **Bound key origin rule for symmetric keys:** | The protocol abort when $P$'s Possession set contains the key $k$ which is bound to principle $Q$, and $Q \neq P$ but this binding is not in $P$'s bindings set which contains legal bindings for $P$. |
| **Bound key origin rules for asymmetric keys:** | If principle $P$ has the binding of an asymmetric key to principle $Q$ in its possession and binding sets and $P$'s possession set contains a message that is encrypted under the inverse key this means that the message comes from $Q$ because it is the only principle that have the inverse key. Otherwise, the protocol aborts. |
| **Conjunction rule:** | When $P$'s Proof set contains the proof that $Q$ is accountable for the conjunction $X$ and $Y$, then it will contain the proof that $Q$ is accountable for $X$ and $Y$, respectively. Contrariwise, the second one is used. |
| **Ciphertext understanding rule:** | When $P$'s Proof set contains the prove that $Q$ is accountable for $\{X\}_k$ and $Q$ holds $k$ then it will contain the prove that $Q$ is accountable for message $X$. |
| **Transfer rule:** | When $P$'s Proof set contains the proof that Trusted Third Party (TTP) is accountable for message $X$, then it will contain the proof that $Q$ holds message $X$. |
| **Timestamp rule (symmetric keys):** | Timestamp rule defined to reason about the message. |
| **Timestamp rule (asymmetric keys):** | Timestamp rule defined to reason about the message. |
| **Authentication Rule:** | This rule adds a new binding to the binding set. Note that its condition will not complete until the comparison action (ACT$\_1$) is executed successfully. $Z$ could be any principle except $P$ or TTP. |
(iii) **MAC Analysis.** The last step is to analyze the security enforcer which is MAC in this case to check security requirements it achieves.

Alice is assumed to be the initiator of the protocol. Therefore, four actions in the BL(A) are executed which add new values to the POSS(A) set:

\[
\text{POSS}(A) = \{ \text{Msg}, K_{AB} \mid (B \equiv K_{AB}) \text{ from } B, \\
\{ \text{Msg}, K_{AB} \}, H(\text{Msg}, K_{AB}) \}. 
\]

So far no inference rules can be applied. Now, actions in BL(B) are executed and this will produce new values that are added to POSS(B) and the Proof(B) sets. Consequently, both integrity and authentication inference rules can be applied:

\[
\text{POSS}(B) = \{ K_{AB} \mid (A \equiv K_{AB}) \text{ from } A, \text{Msg}, \\
H(\text{Msg}, K_{AB}), \{ \text{Msg}, K_{AB} \}, H(\text{Msg}, K_{AB}) \} \
\text{Proof}(B) = \{ \text{Msg}^* \}. 
\]

Since \{Msg\}^* \in Proof(B), we can apply the authentication rule (RL17) and the integrity rule (RL18) as follows.

**Authentication Rule (RL17):**

\[
(A \equiv K_{AB}) \text{ from } A \in \text{POSS}(B) \\
\text{Binding}(B) = \text{Binding}(B) \cup \{ A \equiv K_{AB} \}. 
\]

**Integrity Rule (RL18):**

\[
\{ \text{Msg}^* \in \text{Proof}(B) \\
\text{BEL}(B) = \text{BEL}(B) \cup \{ \text{Msg} \in \text{POSS}(A) \}. 
\]

The equality of the two digests also proves the correctness of the received message which leads Bob to adding a new value to Bob’s Belief set indicating that the received message is the same as the message sent by Alice.

As can be concluded, LBSA has been used to mathematically prove the achievement of authentication and integrity by MAC. The above process is also applicable to any other security enforcer.

### 6.2 Digital Certificate

In hierarchical trust model, digital certificate [14] is usually issued by a certification authority (CA). It contains the user identity, public key (K\(^+\)), CA’s digital signature, and other information. Using the CA’s digital signature, the user can prove the authenticity of the issuer and ensure correct binding between the identity and the public key in the certificate.

We assume that the CA is the initiator of the protocol which will make one certificate for Alice and another for Bob. It sends each certificate after attaching its signature. Bob needs Alice’s public key to send messages to her securely, therefore Alice sends her certificate to Bob then Bob starts the signature verification to ensure correct binding between Alice’s public key and her identity.
Using LBSA to prove the achieved security requirements using the digital certificate, we start by presenting the
enforcer specification in terms of global and local sets and then we performed the analysis part.

(i) Define the Values of the Global Sets. Consider

\[(GS1) \ P = \{CA, Alice, Bob\}\]
\[(GS2) \ R = \{\text{contains the rules defined in Tables 4(a)–4(d)}\}\]
\[(GS3) \ S = \{K_{CA}^-, K_A^+, K_B^+\}\]
\[(GS4) \text{Observers} (K_{CA}^-) = \{CA\}\]
\[(GS4) \text{Observers} (K_A^+) = \{A\}\]
\[(GS4) \text{Observers} (K_B^+) = \{B\}\].

(ii) Define the Initial Values of the Local Sets to each Principal

\[
\text{Principal (CA):} \hspace{1cm} \begin{align*}
\text{(LS1) } & \text{POSS(CA)} = \{K_{CA}^-, K_{CA}^+\} \\
\text{(LS2) } & \text{BEL(CA)} = \{(#(K_{CA}^-), #(K_A^+))\} \\
\text{(LS7) } & \text{Binding(CA)} = \{CA \bowtie K_{CA}^+\} \\
\text{(LS6) } & \text{BL} =
\end{align*}
\]
\[// \text{Generation of Alice's certificate}\]
\[\text{(ACT}_{12}\text{)} \ \text{Apply}(H, cert_A) // \text{digest generation}\]
\[\text{(ACT}_{15}\text{)} \ \text{Apply-asymkey}(H(cert_A), K_{CA}^-)// \text{generation of CA digital signature}\]
\[\text{(ACT}_{7}\text{)} \ \text{Concat}(cert_A, \{H(cert_A)\}^C_{CA})\]
\[\text{(ACT}_{1}\text{)} \ \text{Send}(Alice, cert_A, \{H(cert_A)\}^C_{CA}) \hspace{1cm} // \text{Generation of Bob's certificate}\]
\[\text{(ACT}_{12}\text{)} \ \text{Apply}(H, cert_B)\]
\[\text{(ACT}_{15}\text{)} \ \text{Apply-asymkey}(H(cert_B), K_{CA}^-)\]
\[\text{(ACT}_{7}\text{)} \ \text{Concat}(cert_B, \{H(cert_B)\}^C_{CA})\]
\[\text{(ACT}_{1}\text{)} \ \text{Send}(Bob, cert_B, \{H(cert_B)\}^C_{CA}).\]

\[
\text{Principal (A) (Alice):} \hspace{1cm} \begin{align*}
\text{(LS1) } & \text{POSS(A)} = \{K_A^-, K_A^+, K_{CA}^+\} \\
\text{(LS2) } & \text{BEL(A)} = \{(#(K_A^-), #(K_A^+), #(K_{CA}^-))\} \\
\text{(LS7) } & \text{Binding(CA)} = \{A \bowtie K_A^+, CA \bowtie K_{CA}^+\} \\
\text{(LS6) } & \text{BL} =
\end{align*}
\]
\[\text{(ACT}_{2}\text{)} \ \text{Receive}(CA, cert_A, \{H(cert_A)\}^C_{CA}) \hspace{1cm} (ACT_{1}) \ \text{Send}(Bob, cert_B, \{H(cert_B)\}^C_{CA}) \hspace{1cm} // A\text{ executes these actions when Bob sends its certificate to A}\]

(iii) Digital Certificate Analysis. CA is assumed to be the initiator of the protocol and the first four actions in the
BL(CA) are executed which will add new values to the
POSS(CA) as follows:

\[
\text{POSS(CA)} = \{K_{CA}^-, K_{CA}^+, H(cert_A), \{H(cert_A)\}^C_{CA},
\]
\[\text{cert_A, } \{H(cert_A)\}^C_{CA}\}.\]

The next action to be executed is the first action in BL(A) which will receive the msg and update
the POSS(A) to include the msg content:

\[
\text{POSS(A)} = \{K_A^-, K_A^+, K_{CA}^+, K_{CA}^+, cert_A, \{H(cert_A)\}^C_{CA}\}.\]

Then CA does the rest of the actions which followed by executing the first action in BL(B) and these
actions update the POSS(CA) again and the POSS(B).

\[
\text{POSS(CA)} = \{K_{CA}^-, K_{CA}^+, H(cert_A), \{H(cert_A)\}^C_{CA},
\]
\[\text{cert_A, } \{H(cert_A)\}^C_{CA}, H(cert_B), \{H(cert_B)\}^C_{CA},
\]
\[\text{cert_B, } \{H(cert_B)\}^C_{CA}\}.\]

\[
\text{POSS(B)} = \{K_B^-, K_B^+, K_{CA}^+, cert_B, \{H(cert_B)\}^C_{CA}\}.\]

Table 6: Mappings between protocol services and inference rules/actions.

| Protocol Services                          | Inference rules/actions |
|-------------------------------------------|-------------------------|
| Key management                            | ACT_6, ACT_11, ACT_14, ACT_16, ACT_19, ACT_22, ACT_24, ACT_30, RL_4, RL_5, RL_6, RL_10, RL_11 |
| Nodes interaction and protocol functions | ACT_1, ACT_2, ACT_7, ACT_9, ACT_12, ACT_17, ACT_18, ACT_20, ACT_21, ACT_23, ACT_29, RL_14, ACT_5, ACT_10 |
| Data refreshment                          | RL_14, RL_15, RL_16, RL_18, RL_153, RL_16 |
Now Alice sends her certificate to Bob which is the action A(1) in the BL(A) and the next actions to be executed are B(1)-B(2) which update the POSS(B):

\[
\text{POSS(B) = \{K^-_B, K^+_B, K^+_CA, cert_B, H(cert_B)_{K^-_CA}, cert_A, [H(cert_A)_{K^-_CA}], (K^+_\lambda \land A)\}}
\]

Since \([H(cert_A)_{K^-_CA}] \in \text{POSS(B)}\) we can apply Submessage origin rule (RL7):

\[
\frac{\text{cert}_A, [H(cert_A)_{K^-_CA}] \in \text{POSS(B)}}{(K^+_\lambda \land A) \text{ from CA \in POSS(B)}}. \quad (5)
\]

Which adding a new value to the POSS(B):

\[
\text{POSS(B) = \{K^-_B, K^+_B, K^+_CA, cert_B, H(cert_B)_{K^-_CA}, cert_A, [H(cert_A)_{K^-_CA}], (K^+_\lambda \land A)\}} \text{ from CA}. \]

Finally, executing the action B(3) will add new values to POSS(B) (assuming the comparison action ensures equality) which achieves some rules conditions and these rules can be applied:

\[
\frac{\text{POSS(B) = \{K^-_B, K^+_B, K^+_CA, cert_B, H(cert_B)_{K^-_CA}, cert_A, [H(cert_A)_{K^-_CA}], (K^+_\lambda \land A)\}} \text{ from CA, H(cert_A), H(cert_A), \{cert_A\}^*}.}{(6)}
\]

Since \((K^+_\lambda \land A)\) from CA \in POSS(B), \{cert_A\}^* \in POSS(B) we can apply authentication and integrity rules as follows.

**Authentication Rule (RL17):**

\[
\frac{(K^+_\lambda \land A) \text{ from CA \in POSS(B)}}{\text{Binding(B) = Binding(B) \cup \{(K^+_\lambda \land A)\}^*}}. \quad (6)
\]

**Integrity Rule (RL18):**

\[
\frac{\{\text{cert}_A\}^* \in \text{Proof(B)}}{\text{BEL(B) = BEL(B) \cup \{\text{cert}_A \in \text{Poss(A)}\}}}.
\]

**Nonrepudiation Rule (RL19):**

\[
\frac{\text{H(cert}_A\}_{K^-_CA} \in \text{Poss(B), (K^+_\lambda \land A) \text{ in Binding(B)}}}{\text{(CA \rightarrow cert}_A\) \in \text{Proof(B)}}. \quad (8)
\]

Consequently, applying Authentication Rule (RL17) ensures the authenticity of Alice’s public key. Integrity Rule (RL18) confirms the correctness of Alice’s certificate. Finally, Nonrepudiation Rule (RL19) proves the originality of Alice’s certificate, in other words it proves that Alice’s certificate was issued and signed by the CA.

The purpose of LBSA is to generalize the rules, but the way they are applied depends on the enforcer itself. As shown this section, Rules 17 and 18 are used by both MAC and digital certificates but in different meanings. The authentication rule in MAC ensures the authenticity of the received message, whereas in digital certificate it ensures the authenticity of the public key. The integrity rule is used in MAC to ensure the correctness of the received message whereas in digital certificate it used to ensure the correctness of the certificate’s contents.

### 7. Case Study: Ariadne Protocol

Many secure routing protocols have been proposed in the literature [27–32]. In this section, Ariadne protocol [27], which is a secure, on-demand routing protocol for multihop wireless networks was chosen to illustrate how to use LBSA to specify and analyze secure protocols. Ariadne is based on Timed Efficient Stream Loss-tolerant Authentication (TESLA) [33] which is an efficient broadcast authentication scheme that requires time synchronization. TESLA is commonly used in wireless sensor networks due to its low communication and computation overhead.

Before we start the security verification of Ariadne, we will present an overview of Ariadne route discovery using TESLA. For simplicity, we will use four nodes but what is applied in case of four nodes can be applied to more nodes. In all cases we will have a source, a destination, and a set of intermediate nodes which their number can vary depending on the chosen route. In the case of using more nodes, the change will be in the length of the chain of nodes. The way the authentication and integrity are achieved is defined by the Ariadne protocol itself. In this case study, we mapped Ariadne to LBSA to check Ariadne security correctness.

The source node S begins route instantiation to destination D by broadcasting a route request packet. A and B are intermediate nodes, that is, \(S \rightarrow A \rightarrow B \rightarrow D\). A route reply packet is unicasted by the destination D as a reply to the request packet along the reverse path to the source.

In the route discovery process there are two types of messages REQUEST and REPLY. REQUEST and REPLY messages include the following fields:

- Route REQUEST packet in Ariadne contains eight fields: REQUEST (define the type of the message), initiator address, target address, id (uniquely identifies the request), time interval, hash chain, node list, and MAC list. Note that the last three fields are updated by each intermediate node receiving that request.

Table 7 shows the protocol parameters.

The following steps illustrate how Ariadne route discovery using TESLA works:

\[
S: h_0 = \text{MAC}_{KSD}(\text{REQUEST}, S, D, \text{id}, \text{ti})
\]

\[
S \rightarrow \text{broadcast: (REQUEST, S, D, id, ti, h_0, \{\}, \{\})}
\]

\[
A: h_1 = H(A, h_0)
\]

\[
M_A = \text{MAC}_{KSA}(\text{REQUEST}, S, D, id, ti, h_1, (A), \{\})
\]
7.1. Ariadne Specification. First we will present the assumptions of Ariadne protocol which are as follows.

(i) The source S and destination D share the MAC keys \( K_{SD} \) and \( K_{DS} \).
(ii) Every node has a TESLA one-way key chain.
(iii) All nodes know the authentication key of TESLA one-way key chain of each other node.

(i) Define the Values of the Global Sets. Consider

\[ (GS1) P = \{S, A, B, D\} \]
\[ (GS2) R = \{\text{contains the rules defined in Tables 4(a)–4(d)}\} \]
\[ (GS3) S = \{K_{SD}, K_{DS}, K_{Bu}, K_{Aii}\} \]
\[ (GS4) \text{Observers} \ (K_{SD}) = \{S, D\} \]
\[ (GS4) \text{Observers} \ (K_{Aii}) = \{A, S\} \]
\[ (GS4) \text{Observers} \ (K_{Bu}) = \{B, S\} \]
\[ (GS4) \text{Observers} \ (K_{DS}) = \{D, S\} \]

(ii) Define the Initial Values of the Local Sets for each Principal

**Principal (S):**

\[ (LS1) \text{POSS}(S) = \{K_{DS}, K_{SD}, K_{Aii}, K_{Bu}, (D \times K_{DS}) \}
\]
\[ \text{from } D, (D \times K_{SD}) \text{ from } D, (A \times K_{Aii}) \text{ from } A, \]
\[ (B \times K_{Bu}) \text{ from } B \]

\[ (LS2) \text{BEL}(S) = \{\#(K_{DS}), \#(K_{SD}), \#(K_{Aii}), \#(K_{Bu}), \]
\[ (D \times K_{DS}), (D \times K_{SD}), (A \times K_{Aii}), (B \times K_{Bu})\} \]

\[ (LS6) \text{BL} = \]
\[ (ACT_{7}) \text{Concat } ([\text{REQUEST}, S, D, \text{id, ti}, K_{SD}]) = S(1) \]
\[ (ACT_{12}, ACT_{29}) h_0 \leftarrow \text{Apply}(H, \{\text{REQUEST}, S, D, \text{id, ti, } K_{SD}\}) = S(2) \]
\[ (ACT_{19}) \text{Broadcast } ([\text{REQUEST}, S, D, \text{id, ti, } h_0, \]
\[ (, , )]) = S(3) \]
\[ (ACT_{18}) \text{Receive } (A, \{\text{REPLY, D, S, ti, } (A, B), \]
\[ (M_A, M_B), M_D, (K_{Bu}, K_{Aii})\}) = S(4) \]
\begin{align*}
\text{(ACT}_7\text{)} & \text{ Concat}([\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), K_{DS}]) \rightarrow \text{S}(5) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ C1 } \leftarrow \text{ Apply}(H, [\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), K_{DS}]) \rightarrow \text{S}(6) \\
\text{(ACT}_{21}) & \text{ Comp}(M_D, C1) \rightarrow \text{S}(7) \\
\text{(ACT}_7\text{)} & \text{ Concat}([\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), \{\}, A_i]) \rightarrow \text{S}(8) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ C2 } \leftarrow \text{ Apply}(H, [\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), \{\}, A_i]) \rightarrow \text{S}(9) \\
\text{(ACT}_{21}) & \text{ Comp}(M_A, C2) \rightarrow \text{S}(10) \\
\text{(ACT}_7\text{)} & \text{ Concat}([\text{REQUEST}, S, D, \text{id}, t_i, h2, (A, B), \{\}, B_i]) \rightarrow \text{S}(11) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ C3 } \leftarrow \text{ Apply}(H, [\text{REQUEST}, S, D, \text{id}, t_i, h2, (A, B), \{\}, B_i]) \rightarrow \text{S}(12) \\
\text{(ACT}_{21}) & \text{ Comp}(M_B, C3) \rightarrow \text{S}(13). \\
\end{align*}

\textbf{Principal (A)}:

\begin{itemize}
\item \text{(LS1)} \text{POSS}(A) = \{A_{ti}\}
\item \text{(LS2)} \text{BEL}(A) = \#(A_{ti})
\item \text{(LS7)} \text{Binding}(A) = \{A_{ti} \bowtie t_i\}
\item \text{(LS6)} \text{BL} =
\begin{align*}
\text{(ACT}_7\text{)} & \text{ Receive}([\text{REQUEST}, S, D, \text{id}, t_i, h0, \{\}, \{\})] \rightarrow \text{A}(1) \\
\text{(ACT}_{10}) & \text{ Check-freshness}(t_i) \rightarrow \text{A}(2) \\
\text{(ACT}_{10}) & \text{ Check-request}(S, \text{id}) \rightarrow \text{A}(3) \\
\text{(ACT}_7\text{)} & \text{ Concat}(A, h0) \rightarrow \text{A}(4) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ h1 } \leftarrow \text{ Apply}(H, [A, h0]) \rightarrow \text{A}(5) \\
\text{(ACT}_7\text{)} & \text{ Concat}([\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), \{\}, A_i]) \rightarrow \text{A}(6) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ M_A } \leftarrow \text{ Apply}(H, [\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), \{\}, A_i]) \rightarrow \text{A}(7) \\
\text{(ACT}_{18}) & \text{ Broadcast}([\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), (M_A)]) \rightarrow \text{A}(8) \\
\text{(ACT}_7\text{)} & \text{ Receive}(B,[\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), M_D, (K_{Bi}))]) \rightarrow \text{A}(9) \\
\text{(ACT}_7\text{)} & \text{ Send}(B, [\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), M_D, (\{\}))]) \rightarrow \text{A}(10).
\end{align*}
\end{itemize}

\textbf{Principal (B)}:

\begin{itemize}
\item \text{(LS1)} \text{POSS}(B) = \{B_{ti}\}
\item \text{(LS2)} \text{BEL}(B) = \#(B_{ti})
\item \text{(LS7)} \text{Binding}(B) = \{B_{ti} \bowtie t_i\}
\item \text{(LS6)} \text{BL} =
\begin{align*}
\text{(ACT}_7\text{)} & \text{ Receive}([\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), (M_A)]) \rightarrow \text{B}(1) \\
\text{(ACT}_{20}) & \text{ Check-request}(S, \text{id}) \rightarrow \text{B}(2) \\
\text{(ACT}_{10}) & \text{ Check-freshness}(t_i) \rightarrow \text{B}(3) \\
\text{(ACT}_7\text{)} & \text{ Concat}(B, h1) \rightarrow \text{B}(4) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & h2 \leftarrow \text{ Apply}(H, [B, h1]) \rightarrow \text{B}(5) \\
\end{align*}
\end{itemize}

\textbf{Principal (D)}:

\begin{itemize}
\item \text{(LS1)} \text{POSS}(D) = \{K_{DS}, K_{SD}, (S \bowtie K_{DS}) \text{ from } S, (S \bowtie K_{DS}) \text{ from } S\}
\item \text{(LS2)} \text{BEL}(D) = \#(K_{SD}), \#(K_{SD}) \bowtie S, K_{DS} \bowtie S
\item \text{(LS6)} BL =
\begin{align*}
\text{(ACT}_7\text{)} & \text{ Receive}([\text{REQUEST}, S, D, \text{id}, t_i, h2, (A, B), (M_A, M_B)]) \rightarrow \text{D}(1) \\
\text{(ACT}_{20}) & \text{ Check-request}(S, \text{id}) \rightarrow \text{D}(2) \\
\text{(ACT}_{10}) & \text{ Check-freshness}(t_i) \rightarrow \text{D}(3) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ C1 } \leftarrow \text{ Apply}(H, \text{Concat}(A, \text{Apply}(H, \text{Concat}(B, \text{Apply}(H, \text{Concat}(\text{REQUEST}, S, D, \text{id}, t_i, K_{SD})))))) \rightarrow \text{D}(4) \\
\text{(ACT}_{21}) & \text{ Comp}(h2, C1) \rightarrow \text{D}(5) \\
\text{(ACT}_7\text{)} & \text{ Concat}([\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), K_{DS}]) \rightarrow \text{D}(6) \\
\text{(ACT}_{12}, \text{ ACT}_{29}) & \text{ M_D } \leftarrow \text{ Apply}(H, [\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), K_{DS}]) \rightarrow \text{D}(7) \\
\text{(ACT}_7\text{)} & \text{ Send}(B, [\text{REPLY}, D, S, \text{id}, t_i, (A, B), (M_A, M_B), M_D, (\{\}))]) \rightarrow \text{D}(8).
\end{align*}
\end{itemize}

7.2 Ariadne Analysis. Node S is the initiator of the protocol and its first three actions will be executed which result in new values to be added to the \text{POSS}(S) set:

\text{POSS}(S) := \text{POSS}(S) \cup \{(\text{REQUEST}, S, D, \text{id}, t_i, K_{SD}\}, \{h_0 = H([\text{REQUEST}, S, D, \text{id}, t_i, K_{SD}])\}.

Now we assume that node A receives the request so first eight actions in BL(A) will be executed and the result is

\text{POSS}(A) = \text{POSS}(A) \cup \{(\text{REQUEST}, S, D, \text{id}, t_i, h0, \{\}, \{\}, A_i, h1 = H(A_i)) \}

\text{M_A} = H([\text{REQUEST}, S, D, \text{id}, t_i, h1, (A), \{\}).A_i))

\text{BEL}(A) = \text{BEL}(A) \cup \#(t_i).

Now B(1–8) will be executed and the result is

\text{POSS}(B) = \text{POSS}(B) \cup \{(\text{REQUEST}, S, D, \text{id}, t_i, h2, (A), (M_A), B, h1, h2 = H(B, h1), (\text{REQUEST}, S, D, \text{id}, t_i, h2, (A), \{\}).B_i, M_B = H([\text{REQUEST}, S, D, \text{id}, t_i, h2, (A), \{\}).B_i)) \}

\text{BEL}(B) = \text{BEL}(B) \cup \#(t_i).
Now the request reaches its target which is node D and its BL(1–5) actions will be executed and the result will be as follows:

\[
\text{POSS}(D) = \text{POSS}(D) \cup \{(\text{REQUEST}, S, D, \text{id}, \text{ti}, h_2, (A, B), (M_A, M_B)), C1 = H(A \cdot B), (\text{REQUEST}, S, D, \text{id}, \text{ti}, K_{SD}), \{\text{REPLAY}, D, S, \text{id}, \text{ti}, (A, B), (M_A, M_B)\}\}
\]

\[
\text{BEL}(D) = \text{BEL}(D) \cup \{(#(t_i))\}
\]

\[
\text{Proof}(D) = \text{Proof}(D) \cup \{h_2\}^*
\]

Since \(h_2\)^* \in \text{Proof}(D) we can apply integrity rule RL(18): \(h_2\)^* \in \text{Proof}(D)/ \text{BEL}(D) = \text{BEL}(D) \cup \{\{\text{REQUEST}, S, D, \text{id}, \text{ti}, h_2, (), ()\}\in \text{Poss}(S)\}.

After the execution of D(7) and D(8), the POSS set will be updated as follows:

\[
\text{POSS}(D) = \text{POSS}(D) \cup \{\text{REQUEST}, S, D, \text{id}, \text{ti}, h_2, (A, B), (M_A, M_B)), C1 = H(A \cdot B), (\text{REQUEST}, S, D, \text{id}, \text{ti}, K_{SD}), \{\text{REPLAY}, D, S, \text{id}, \text{ti}, (A, B), (M_A, M_B)\}\}
\]

Therefore, the target (destination) node makes sure that the request is from node S and it will send a reply message using the reverse path since the comparison action confirms equality. Each intermediate node \((A \text{ and } B \text{ in our example})\) will receive the reply, adds its key to the key list, and forwards it to the next node until it reaches the source. The destination will authenticate each intermediate node using the same process as the target node did.

When the target node S receives the REPLY the rest of its actions will be executed to ensure the validity of the target's MAC, and each MAC in the MAC list. Node S local sets will be updated as follows:

\[
\text{POSS}(S) = \{K_{DS}, K_{SD}, K_{AS}, K_{BS}, (A \cong K_{AS}), (A \cong K_{AS})\text{ from }A_1, (B \cong K_{BS})\text{ from }B, \{\text{REPLY}, D, S, \text{ti}, (A, B), (M_A, M_B)), (K_{BS}K_{AS}), \{\text{REPLY}, D, S, \text{id}, \text{ti}, (A, B), (M_A, M_B)\}, \{\text{REQUEST}, S, D, \text{id}, \text{ti}, h_1, (A, ()), ()), A_1, \{\text{REQUEST}, S, D, \text{id}, \text{ti}, h_2, (A, B), ()\}\text{ from }B_1\}
\]

\[
\text{Proof}(S) = \text{Proof}(S) \cup \{\{M_D\}^*, \{M_B\}^*, \{M_A\}^*\}
\]

Now we can apply the authentication rule (RL17) for each entry in the proof set.

**Authentication Rule (RL17):**

\[
\begin{align*}
(D \cong K_{DS}) & \text{ from } D \in \text{POSS}(S) \\
\text{Binding}(S) = & \text{Binding}(S) \cup \{\{D \cong K_{DS}\}\}^*
\end{align*}
\]

\[
\begin{align*}
(D \cong K_{BD}) & \text{ from } D \in \text{POSS}(S) \\
\text{Binding}(S) = & \text{Binding}(S) \cup \{\{D \cong K_{DS}\}\}^*
\end{align*}
\]

\[
\begin{align*}
(B \cong K_{BS}) & \text{ from } B \in \text{POSS}(S) \\
\text{Binding}(S) = & \text{Binding}(S) \cup \{\{B \cong K_{BS}\}\}^*
\end{align*}
\]

\[
\begin{align*}
(A \cong K_{AB}) & \text{ from } A \in \text{POSS}(S) \\
\text{Binding}(S) = & \text{Binding}(S) \cup \{\{A \cong K_{AB}\}\}^*
\end{align*}
\]

Also we can apply the integrity rule (RL18) for each entry in the proof set.

**Integrity Rule (RL18):**

\[
\begin{align*}
\{M_D\}^* & \in \text{Proof}(S) \times (\text{BEL}(S) : = \text{BEL}(S)) \\
& \cup \{\{\text{REPLY}, D, S, \text{ti}, (A, B), (M_A, M_B), M_D, ()\}\text{ from }A_1, \{\text{REQUEST}, S, D, \text{id}, \text{ti}, h_2, (A, B), ()\}\text{ from }B_1\}
\end{align*}
\]

Since we applied the authentication and the integrity rules successfully on the target's MAC and on each MAC in the MAC list, it can be concluded that the reply is valid and authenticated; consequently node S can accept it. Therefore, Ariadne achieves both the authentication and the integrity requirements which detect attackers when they attempt to impersonate the message's sender or modify its content.

Flaws in this protocol lie in the assumptions of the authors that may affect the usability of the protocol. Ariadne assumed the presence of a working public key infrastructure (PKI) or certificate authority (CA). Consequently, key management services are assumed to be already existed but are not provided by Ariadne. Also, Ariadne did not take the possibility of having internal attackers into consideration and therefore, no IDS are applied to detect them.

Based on the above assumptions, actions such as ACT_4, ACT_11, ACT_13, ACT_14, ACT_16, ACT_19, ACT_22, ACT_24, ACT_25, ACT_26, ACT_27, and ACT_28 cannot be activated and rules like R4, R5, R9, R10, and R11 cannot be applied. Based on this observation, LBSA can detect the absence of key management system and IDS in Ariadne. As a result, key management services will fail and intruders cannot be detected.
8. Conclusions and Future Work

The success of applications over multihop communication networks principally depends on the achievement of security requirements. Many secure protocols are proposed, consequently, there should be a reliable way to test if these protocols satisfy the security requirements as their designers claim.

Previous attempts to utilize logic in analyzing security have either focused on a specific requirement or a specific protocol. In this paper, we proposed a logic-based security architecture (LBSA) to specify and analyze any security requirement in any system providing multihop communication using logic. Any system or protocol claiming security can be mapped into our architecture in other words it must be specified using various global/local sets and actions, then it can be analyzed by applying different rules. LBSA provides a formal way to analyze security enforcers and security protocols instead of depending only on the simulation tools which are arguable by many researchers in terms of implementation accuracy.

Additionally, using simulation tools usually covers a small set of scenarios whereas using LBSA more cases can be covered that usually cannot be covered exhaustively by simulation tools.

This paper also illustrated how LBSA can be used to verify security enforcers by applying it to two well-known enforcers which are MAC and digital certificate. In addition to that, a case study for a secure routing protocol used in ad hoc and sensor networks named Ariadne was also evaluated in terms of security using LBSA.

To the best of our knowledge, the proposed security architecture covers the most important enforcers. Further development of new enforcers in the future may simply need other sets, actions, and rules to be added. Also, LBSA could be extended by considering new rules and actions for the Interaction between Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). Moreover, LBSA can utilize the computational Intelligence tools in analyzing the security.

References

[1] ITU-T Recommendation X. 800, “Security architecture for Open Systems Interconnection for CCITT applications,” 1991.
[2] ITU-T Recommendation X. 805, “Security architecture for systems providing end-to-end communication,” 2003.
[3] A. D. Rubin and P. Honeyman, “Nonmonotonic cryptographic protocols,” in Proceedings Computer Security Foundations Workshop, pp. 100–116, June 1994.
[4] A. D. Rubin, “Extending NCP for protocols using public keys,” Mobile Networks and Applications, vol. 2, no. 3, pp. 227–241, 1997.
[5] Y. Xu and X. Xie, “Security analysis of routing protocol for MANET based on extended Rubin logic,” in Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC ’08), pp. 1326–1331, Sanya, China, April 2008.
[6] Y. Xu and X. Xie, “Extending Rubin logic for electronic commerce protocols,” in Proceedings of the 2nd International Conference on Anti-counterfeiting, Security and Identification (ASID ’08), pp. 448–451, Guiyang, China, August 2008.
[7] Y. Xu and X. Xie, “Analysis of electronic commerce protocols based on extended Rubin logic,” in Proceedings of the 9th International Conference for Young Computer Scientists (ICYCYS ’08), pp. 2079–2084, Hunan, China, November 2008.
[8] Y. Xu and X. Xie, “Analysis of authentication protocols based on Rubin logic,” in Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM ’08), Dalian, China, October 2008.
[9] A. D. Rubin and P. Honeyman, “Long running jobs in an authenticated environment,” in Proceedings of the USENIX Security Conference IV, pp. 19–28, October 1993.
[10] R. M. Needham and M. D. Schroeder, “Using encryption for authentication in large networks of computers,” Communications of the ACM, vol. 21, no. 12, pp. 993–999, 1978.
[11] K. Sanzgiri, D. LaFlamme, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, “Authenticated routing for ad hoc networks,” IEEE Journal on Selected Areas in Communications, vol. 23, no. 3, pp. 598–610, 2005.
[12] J. Zhou and D. Gollmann, “Fair non-repudiation protocol,” in Proceedings of the 17th IEEE Symposium on Security and Privacy, pp. 55–61, Oakland, Calif, USA, May 1996.
[13] M. Satyanarayanan, “Integrating security in a large distributed system,” ACM Transactions on Computer Systems, vol. 7, no. 3, pp. 247–280, 1989.
[14] ITU-T Recommendation X. 509, “Information technology—Open System Interconnection—The Directory: public-key and attribute certificate framework,” 2008.
[15] J. Weiss, “Message digest, message authentication codes and digital signature,” in Jave Cryptography Extensions, pp. 101–118, 2004.
[16] J. Zhou and K. Y. Lam, “Securing digital signatures for non-repudiation,” Computer Communications, vol. 22, no. 8, pp. 710–716, 1999.
[17] Z. Shao, “Certificate-based fair exchange protocol of signatures from pairings,” Computer Networks, vol. 52, no. 16, pp. 3075–3084, 2008.
[18] C. H. Lima and P. J. Lee, “Korean certificate-based digital signature algorithm,” Computers and Electrical Engineering, vol. 25, no. 4, pp. 249–265, 1999.
[19] S. Madria and J. Yin, “SeRWA: a secure routing protocol against wormhole attacks in sensor networks,” Ad Hoc Networks, vol. 7, no. 6, pp. 1051–1063, 2009.
[20] S. Ozdemir and H. Çam, “Integration of false data detection with data aggregation and confidential transmission in wireless sensor networks,” IEEE/ACM Transactions on Networking, vol. 18, no. 3, pp. 736–749, 2010.
[21] E. Stavrou and A. Pitsillides, “A survey on secure multipath routing protocols in WSNs,” Computer Networks, vol. 54, no. 13, pp. 2215–2238, 2010.
[22] O. Delgado-Mohatar, A. Fúster-Sabater, and J. M. Sierra, “A light-weight authentication scheme for wireless sensor networks,” Ad Hoc Networks, vol. 9, no. 5, pp. 727–735, 2011.
[23] I. Almomani and M. Saadeh, “Security model for tree-based routing in wireless sensor networks: structure and evaluation,” KSII Transactions on Internet and Information Systems, vol. 6, no. 4, pp. 1223–1247, 2012.
[24] K. Bicakci, I. E. Bagci, and B. Tavli, “Communication/computation tradeoffs for prolonging network lifetime in wireless sensor networks: the case of digital signatures,” Information Sciences, vol. 188, pp. 44–63, 2012.
[25] X. Fan and G. Gong, “Accelerating signature-based broadcast authentication for wireless sensor networks,” Ad Hoc Networks, vol. 10, no. 4, pp. 723–736, 2012.

[26] ISO/IEC 9797-2, “Information technology—security techniques—Message Authentication Code (MACs)—Part 2: mechanisms using a dedicated hash function.”

[27] Y. C. Hu, A. Perrig, and D. B. Johnson, “Ariadne: a secure on-demand routing protocol for ad hoc networks,” Wireless Networks, vol. 11, no. 1-2, pp. 21–38, 2005.

[28] R. Mavropodi, P. Kotzanikolaou, and C. Douligeris, “SecMR—a secure multipath routing protocol for ad hoc networks,” Ad Hoc Networks, vol. 5, no. 1, pp. 87–99, 2007.

[29] J. Kim and G. Tsudik, “SRDP: secure route discovery for dynamic source routing in MANETs,” Ad Hoc Networks, vol. 7, no. 6, pp. 1097–1109, 2009.

[30] S. Gupte and M. Singhal, “Secure routing in mobile wireless ad hoc networks,” Ad Hoc Networks, vol. 1, no. 1, pp. 151–174, 2003.

[31] K. K. Chauhan, A. K. S. Sanger, and V. S. Kushwah, “Securing on-demand source routing in MANETs,” in Proceedings of the 2nd International Conference on Computer and Network Technology (ICCNT’10), pp. 294–297, Bangkok, Thailand, April 2010.

[32] S. G. Yoo, K. Y. Park, and J. Kim, “A security-performance-balanced user authentication scheme for wireless sensor networks,” International Journal of Distributed Sensor Networks, vol. 2012, Article ID 382810, 11 pages, 2012.

[33] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, “The TESLA broadcast authentication protocol,” CryptoBytes, RSA Laboratories, vol. 5, no.2, 2002.
