Formal Analysis of Security Protocols with Movement

Andrew Cook, Luca Viganò*

Department of Informatics, King's College London, London, UK

Abstract

Standard approaches to the formal and automated analysis of security protocols typically do not consider explicitly the fact that certain protocols require the agents to move to specific locations in order to send or receive particular messages. In this paper, we formalise a formal and automated approach in which security protocols with explicit agent movement are analysed in the presence of a Dolev-Yao attacker. We define movements as timed processes in an applied pi-calculus that allows us to build traces that show how the location and the movements of the agents and of the attacker significantly affect the execution of a protocol as well as the execution of possible attacks. We have automated our approach by using the UPPAAL tool. As proof-of-concept, we show our approach in action by analysing several security properties of a concrete case study, the SegCom protocol for vehicular ad-hoc networks.

Keywords

Security protocols, Formal methods, Movement, UPPAAL

1. Introduction

A security protocol is an exchange of messages between two or more agents, where the messages may contain information encrypted with pre-shared keys to achieve confidentiality, signed with privately kept keys for purposes of proof of origin of the message and authentication, or hashed to prevent message tampering. A number of techniques and tools have been developed to carry out symbolic analyses of the actual security of the proposed protocols in the presence of an active attacker, such as [1, 2, 3, 4, 5] to name just a few. The model proposed by Dolev and Yao in [6] has become the standard attacker model in symbolic analysis of security protocols, i.e., when one wants to consider a dishonest agent that can do anything except break cryptography, which is assumed to be perfect. However, the models adopted for honest and dishonest agents in these approaches do not consider explicitly the fact that certain protocols require the agents to move to specific locations in order to send or receive particular messages. In other words, an agent may only be able to send or receive a message if it has moved in a specific and correct way, otherwise the protocol is un-executable. This suggests that location and movement of the agents should not only be included in the protocol specification but also be formalised in much the same way that sending and receiving messages is formalised. If we wish to consider movement and location to be a fundamental aspect of a protocol, then we must consider how an attack can be influenced by its inclusion.

The main contribution of this paper is the development of a formal and automated approach

ITASEC 2023: The Italian Conference on CyberSecurity, May 03–05, 2023, Bari, Italy

*Corresponding author.

andrew.cook@kcl.ac.uk (A. Cook); luca.vigano@kcl.ac.uk (L. Viganò)

in which security protocols with explicit agent movement are analysed in the presence of a Dolev-Yao attacker. We define a two-dimensional environment consisting of both static and mobile agents, with a specific range of communication, which means, for instance, that a mobile agent must move into the communication range of a static agent in order for them to be able to exchange messages. We define movements as timed processes in an applied pi-calculus that allows us to build traces that show how the location and the movements of the agents and of the attacker significantly affect not only the execution of a protocol but also the execution of the attacks the protocol suffers from (if any, of course). We have automated our approach by using UPPAAL, a tool for the modelling, validation, and verification of inter-communicating systems. As proof-of-concept, we show our calculus and its UPPAAL implementation in action by analysing security properties of the *SegCom protocol* [7] for vehicular ad-hoc networks, which involves mutual authentication between mobile vehicles and static road-side units.

We proceed as follows. The calculus that we give in § 2 refines and extends the preliminary version that we gave in [8]. In particular, we strengthen the attacker model. In § 3, we implement the calculus in UPPAAL and describe how we capture message passing, movement, cryptography, and the attacker. In § 4, we show our approach in action, illustrating how it allows us to search for and find attacks in our case study. In § 5, we discuss related work. We conclude in § 6 by discussing the power and limitations of our approach, as well as future work.

2. A Calculus of Movement

In this section, we refine and extend the preliminary calculus that we gave in [8]. For the sake of space, we only summarise the main points and highlight the extensions this paper provides. We do not give the full details of the calculus, but only the rules that are relevant in this paper; the full calculus, along with detailed explanations, can be found in [9]. For brevity, we also omit standard applied-pi calculus rules for synchronisation and parallel composition of processes.

Our approach is independent of the specific language of messages, but for concreteness we consider the standard operators for: *message pairing* (m_1, m_2), or simply m_1, m_2; *symmetric encryption* $\{\!|m_1|\!\}_{m_2}$; and *asymmetric encryption* $\{m_1\}_{m_2}$. Other operators (e.g., hashing) can be added easily. We use the standard message manipulation rules given below, where $\mathcal{DY}(M)$ is the set of terms that a *Dolev-Yao attacker* [6] can construct out of the knowledge M.

$$\frac{m \in M}{m \in \mathcal{DY}(M)}\ G_{axiom} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{(m_1, m_2) \in \mathcal{DY}(M)}\ G_{pair} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{\!|m_1|\!\}_{m_2} \in \mathcal{DY}(M)}\ G_{scrypt}$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{m_1\}_{m_2} \in \mathcal{DY}(M)}\ G_{crypt} \qquad \frac{(m_1, m_2) \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\ A_{pair_i} \qquad \frac{\{\!|m_1|\!\}_{m_2} \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\ A_{scrypt}$$

$$\frac{\{m_1\}_{m_2} \in \mathcal{DY}(M) \quad inv(m_2) \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\ A_{crypt} \qquad \frac{\{m_1\}_{inv(m_2)} \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\ A^{inv}_{crypt}$$

The notion of time is formalised by a *global, linear clock*, which is a global variable t that annotates each process in the calculus, and is incremented by the function $inc(t)$. Sending/receiving messages takes no time, but movement of the protocol agents *does* make time pass (more on this later). The *environment* in which agents move is modelled as a two-dimensional affine plane of *nodes*, which are static agents with a finite circular range of communication that exchange messages with mobile, point-like entities that are called *hubs*.

$$\frac{}{x(m).\mathsf{n}_i^t \xrightarrow{x(m)} \mathsf{n}_i^t}\ \mathit{Read}_{\mathsf{n}} \qquad \frac{m \in \mathcal{DY}(K_x^t)}{\overline{x}\langle m\rangle.\mathsf{n}_i^t \xrightarrow{\overline{x}\langle m\rangle} \mathsf{n}_i^t}\ \mathit{Write}_{\mathsf{n}}$$

$$\frac{x \in E_{\mathsf{h}_i}^t}{x(m).\mathsf{h}_i^t \xrightarrow{x(m)} \mathsf{h}_i^t}\ \mathit{Read}_{\mathsf{h}} \qquad \frac{m \in \mathcal{DY}(K_{\mathsf{h}_i}^t) \quad x \in E_{\mathsf{h}_i}^t}{\overline{x}\langle m\rangle.\mathsf{h}_i^t \xrightarrow{\overline{x}\langle m\rangle} \mathsf{h}_i^t}\ \mathit{Write}_{\mathsf{h}}$$

$$\frac{x(m).\mathsf{p}^t \xrightarrow{x(m)} \mathsf{p}^t}{\{m\} \cup K_{\mathsf{p}}^t}\ \mathit{ReadMsg} \qquad \frac{\nu(m).\mathsf{p}^t \xrightarrow{\nu(m)} \mathsf{p}^t}{\{m\} \cup K_{\mathsf{p}}^t}\ \mathit{FreshMsg}$$

Figure 1: Message-Passing Rules for Nodes ($\mathit{Read}_{\mathsf{n}}$ and $\mathit{Write}_{\mathsf{n}}$) and Hubs ($\mathit{Read}_{\mathsf{h}}$ and $\mathit{Write}_{\mathsf{h}}$), and Message-Passing Bridge Rules ($\mathit{ReadMsg}$ and $\mathit{FreshMsg}$)

Nodes in our environment are circles of uniform radius, which are either far apart or overlapping with each other.[1] Each node has a *name* and the radius models the "range of communication" of the node. Then, an *environment* Env is a pair (ϕ, O), where ϕ is a set of *node names* and O is a relation describing the *overlaps* between nodes. Environments are *well formed* if for $O \subseteq \phi \times \phi$ and for all $(n_1, n_2) \in O$, $n_1 \neq n_2$ and $(n_2, n_1) \notin O$ and $|O| < 5$ and Env is planar.[2]

A node's *trace* is described by the behaviour of a process n. In particular, n_i is a node process, where $0 \leq i < |\phi|$ and $|\phi|$ is the size of ϕ. For a finite set of nodes ϕ, there is a finite set of processes $\{\mathsf{n}_0, \mathsf{n}_1, \ldots\}$, where $|\{\mathsf{n}_0, \mathsf{n}_1, \ldots\}| = |\phi|$ and $0 \leq i, j \leq |\{\mathsf{n}_0, \mathsf{n}_1, \ldots\}|$, described by the grammar $\mathsf{n}_i, \mathsf{n}_j ::= U(m).\mathsf{n}_i \mid \overline{U}\langle V\rangle.\mathsf{n}_i \mid \overline{U}\langle V\rangle.\mathsf{n}_i \mid \mathsf{n}_i \parallel \mathsf{n}_j \mid \tau.\mathsf{n}_i \mid 0$, which respectively represent the actions of input, output, composition, internal transition, and halt.

Each protocol agent x possesses a *message knowledge set* K_x^t that represents what messages x knows at global time t based on the messages that it knows initially, has received, or has freshly generated. Nodes are able to send/receive messages at will, on their default channel name. We model this by means of the rules $\mathit{Read}_{\mathsf{n}}$ and $\mathit{Write}_{\mathsf{n}}$ in Figure 1 (again, we point to [9] for the full set of the rules of the calculus).

Hubs are point-like entities in the plane that can move in and out of the ranges of nodes. If a hub is in range of a node n, then it is able to exchange messages with n. If a hub is out of range of n, then to exchange messages with n it must move. If a hub is in the overlapping region of two nodes, then it is able to exchange messages with both nodes without moving.

A hub's *trace* is described by the behaviour of a process h. In particular, h_i is a process of a hub, where $0 \leq i \leq |\mathcal{H}|$ with $\mathcal{H}$ the set of hub processes. The hub processes follow the same applied-pi calculus grammar as the one for node processes. In addition to its message knowledge set, each hub h_i possesses a *location set* $E_{\mathsf{h}_i}^t$ that describes its location at time t: if a hub h enters a node n, the channel name of n is added to h's location set; upon leaving, the channel name of n is removed from h's location set. The following rules for movement define

[1] We impose some sensible restrictions on a node's location: a node cannot "contain" another node (i.e., a smaller node inside a larger node), and only two nodes may *mutually* overlap (i.e., three or more nodes may not all mutually overlap, although a node may overlap with more than one other node). Nodes are also restricted from being positioned directly on top of another, and two nodes may not be just touching at their circumferences.

[2] Environments can be thought of as planar graphs, where the graph nodes represent *our* nodes and edges represent the overlaps. A simple observation is that a complete graph of five nodes K_5 is non-planar.

how a single hub's location set is updated when the hub moves into and out of nodes. These rules are of the form $\frac{J_1}{C}$ or $\frac{J_1 \quad J_2}{C}$, where J_1 is a judgement on the location set of a hub $E^t_{\mathsf{h}_i}$, J_2 is a judgement about the structure of Env, and the conclusion C is an updated location set $E^{inc(t)}_{\mathsf{h}_i}$.

$$\frac{E^t_{\mathsf{h}_i} = \{\} \quad x \in \phi}{E^{inc(t)}_{\mathsf{h}_i} = \{x\}}\ A_{single} \qquad \frac{E^t_{\mathsf{h}_i} = \{x\}}{E^{inc(t)}_{\mathsf{h}_i} = \{\}}\ D_{single} \qquad \frac{E^t_{\mathsf{h}_i} = \{\} \quad (x,y) \in O}{E^{inc(t)}_{\mathsf{h}_i} = \{x,y\}}\ A_{double}$$

$$\frac{E^t_{\mathsf{h}_i} = \{x,y\}}{E^{inc(t)}_{\mathsf{h}_i} = \{\}}\ D_{double} \qquad \frac{E^t_{\mathsf{h}_i} = \{x\} \quad (x,y) \in O}{E^{inc(t)}_{\mathsf{h}_i} = \{x,y\}}\ OA_{single} \qquad \frac{E^t_{\mathsf{h}_i} = \{x,y\} \quad (x,y) \in O}{E^{inc(t)}_{\mathsf{h}_i} = \{x\}}\ OD_{single}$$

Rule	Description
A_{single} / D_{single}	Into a single node / out of a single node.
A_{double} / D_{double}	Into the overlapping region of two nodes / out of overlapping region.
OA_{single}	When in one node that overlaps with another, moving into the overlap.
OD_{single}	When in an an overlap of two nodes, moving to be just inside one.

The rules $Read_{\mathsf{h}}$ and $Write_{\mathsf{h}}$ in Figure 1 allow a hub to send/receive messages, whereas the bridge rules allow the processes to traverse from the world of the implicit to the explicit. They bridge between *any* node/hub/attacker process denoted generically as p and the Dolev-Yao knowledge. $ReadMsg$ takes the premise that a process p has read a message m, and p can then add m to its message knowledge set K^t_{p}. $FreshMsg$ takes as premise the standard creation of a fresh m using the ν operator, and concludes by adding m to p's message knowledge set K^t_{p}.

The attacker a is a dishonest agent that can assume the role of a node or a hub at will. It is a Dolev-Yao attacker that we have extended with movement: it is capable of creating and sending messages by way of the Dolev-Yao message rules, as well as receiving messages either directly on its channel name or by eavesdropping messages sent on another channel only if it has moved within range. The process of the attacker as a inherits all the same capabilities of honest nodes and hubs, and is in possession of a message knowledge set K^t_{a} and location set E^t_{a}.

3. Modelling Security Protocols in UPPAAL

The following techniques capture the capabilities of the agents, and describe the methods in which the communication rules, cryptography, and movement abilities given by our calculus in § 2 are embedded into UPPAAL.

3.1. Message Passing

We begin by embedding in UPPAAL the reading and writing rules from Figure 1. We embed the $FreshMsg$ rule through a function $fresh()$ that returns a single-use typed integer. We concatenate message components through bitwise operations as in [10, 11]. Messages, and components of messages, are conventionally [10, 11] defined as integers of a fixed length of bits in UPPAAL, as UPPAAL calls for typed models. As such, in order for one agent to send a message to another agent, the former writes an integer to a global variable named "message", which another agent may then read. The maximum length of any message is 15 bits. The way that

we type protocol messages is a novel, admittedly minor, contribution; we define integer *ranges* that represent different types of message components in a protocol. To model time-negligible communication, a sender and receiver *synchronise* on an UPPAAL *channel*. Figures 2 and 3 (more details on these models are given in § 4) show an example in which the initiator writes a signed certificate to the global $message$ on the channel $node1$, and the receiver reads this message by synchronising with the initiator on the same channel at the same time.

3.2. Movement

Movements are simple to model in UPPAAL (which is indeed part of the reason why we have chosen to use UPPAAL). We create a model that captures the layout of the environment. For all agents which are hubs, the environment is instantiated with boolean variables that monitor that specific hub's location set. These boolean variables are true or false if the hub is in that location or not. As such, any number of hubs may be instantiated to move around the environment and can freely move in and out of nodes. Each node in the environment is modelled as an UPPAAL model state. If a hub is not in any node, it is modelled as being in *no man's land*, written in UPPAAL as *NML*. An overlapping region of two nodes is itself a unique location and therefore an UPPAAL state. For instance, if two nodes named n_1 and n_2 overlapped, the overlap region is a separate state $n_1\ o\ n_2$ (n_1 overlap n_2), and so on.

3.3. Embedding Cryptography and the Attacker Model in UPPAAL

As in [11], cryptographic operations in UPPAAL are carried out by means of a pair of arrays that all agents are permitted to read and write to. The arrays allow any number of agents to perform encryption and decryption, sign messages, and verify signatures. Other cryptographic operations such as hashing can be added whenever necessary.

The two advantages of embedding cryptography through publicly available functions are: (1) that an arbitrary number of agents can be given cryptographic functionality (similarly to movement), (2) that the underlying mathematics of encryption schemes are abstracted away, which are assumed to be unbreakable by the attacker during analysis and verification regardless.

As mentioned in § 2, the attacker can assume the role of a hub or node at will (inheriting the capabilities of both). The attacker is made up of any number of collaborating *attacking agents* instantiated as hubs, and a singular *attacker brain* which captures the Dolev-Yao rules. The attacker brain is based on the attacker model given in [11] combined with elements of [10], with some further restrictions, optimisations, and modifications of our own. We carefully restrict the attacker brain in UPPAAL so that the automated analysis of the protocol will terminate:

1. *Typing*: Every message component is properly typed.
2. *Message Length*: The maximum length of a message that the attacker brain can construct for a given protocol is bounded.
3. *Shallow Cryptography*: The attacker brain is only allowed to perform some cryptographic technique (e.g., en-/decrypting) *each* only once, every time it sends a message.
4. *Selective Cryptography*: The attacker brain should not be able to use cryptographic techniques that have no relevance to a given protocol.

For each protocol that we wish to analyse in UPPAAL, the attacker brain will need to be constructed in UPPAAL separately for that protocol, as the data involved vary from protocol to protocol. The main novelty of our implementation lies in the ability of the attacker to move, by means of novel attacker agents that all interface with an attacker brain. Additionally, we have optimised the attacker brain further in the following ways:

- The attacker itself can initiate a protocol with an agent.
- Choosing a key and carrying out encryption is the last thing the attacker brain performs before a message is sent.
- Creating a nonce is carried out only during message construction.
- Whenever an encrypted block is received it must either be decrypted and analysed straight away, replayed immediately, or a new message must be constructed.
- "Blocks" are defined to be messages to which no more message components can be added.
- Once message construction has begun, a full block must be built from current knowledge.
- Once a block has been built, it must be sent (encrypted or otherwise). This prevents wasteful actions of the attacker and cyclical construction and discarding of messages in the transition system.

Our modifications allow the least amount of branching paths that the attacker brain transition system is faced with upon receiving a message, to curb the state space explosion. Cycles in the transition system are avoided wherever possible to give the attacker brain a more systematic approach to follow, and wherever choices must be made by the attacker brain, these have been reduced wherever possible in line with the above restrictions.

4. Case Study: The SegCom Protocol

The *SegCom protocol* proposed by Verma and Huang in [7] is a three-phase security protocol between mobile vehicles and *road-side units (RSUs)*. The vehicles are mobile agents, whereas the RSUs are static agents that, critically, have a finite range of communication. Each vehicle and RSU possess a public-private key pair, allowing asymmetric encryption and signing, as well as certificates that they acquire beforehand from a trusted third party. Moreover, SegCom calls for ID-based encryption. In practice, the ID of an RSU, when used as part of ID-based cryptography, is its public key. This type of network is a *Vehicular Ad-Hoc Network (VANET)*, and movement of the vehicles is a defining feature of the network. The highly dynamic nature of the network topology, the short time in which agents may be in range of communication, and the notion of trajectory in the movement make VANETs an interesting case study.

The SegCom protocol consists of three phases: (1) "Primary Authentication", (2) "Group formation for V2V Communication", (3) "Successive Authentication". We focus here on the primary phase of authentication between a mobile vehicle V_1 and an RSU_1 to establish a shared symmetric key, for further (more efficient) secure communication. First, V_1 must move in range of RSU_1 or no messages can be sent or received, and then:

- RSU_1 sends its certificate $Cert_{RSU_1}$ in clear-text upon V_1 entering the RSU's range.
- V_1 validates $Cert_{RSU_1}$ and then sends to RSU_1 the message $MV1 = \{Cert_{V_1}, SK_{V_1,RSU_1}, \{SK_{V_1,RSU_1}\}_{SK_{V_1}}\}_{ID_{RSU_1}}$, where $Cert_{V_1}$ is V_1's certificate, SK_{V_1,RSU_1} is a freshly generated key shared by V_1 and RSU_1, $\{SK_{V_1,RSU_1}\}_{SK_{V_1}}$ is the same key

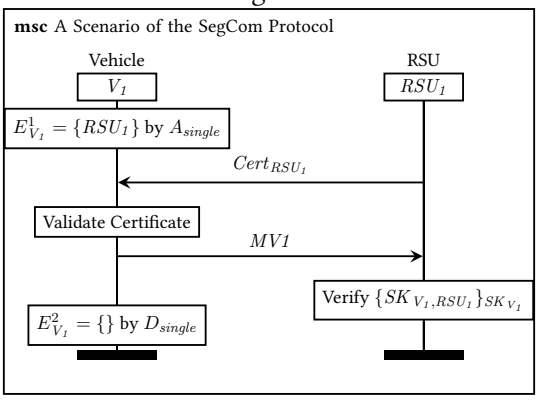
signed by V_1 using its secret key, and ID_{RSU_1} is the public key of RSU_1.

- RSU_1 verifies the signature on the message using V_1's public key (obtained from the certificate, although for simplicity in our implementation the RSU will know the vehicle's public key already). RSU_1 can then be sure that the shared key SK_{V_1,RSU_1} has been created by V_1 and so can be used for subsequent secure communication.
- Finally, as V_1 is moving, eventually it leaves the communication range of RSU_1 and so the key SK_{V_1,RSU_1} is revoked by RSU_1, meaning that V_1 will need to re-certify itself with that RSU if it enters its range again.

Verma and Huang [7] assume that an attacker can deploy adversarial RSUs. In our approach, the attacker has much more power: it can play the role of a mobile hub or static node, depending on the scenario. In this sense, in our analysis, vehicles and RSUs can be honest or dishonest.

4.1. The Protocol in Our Calculus

We formalise the specification of SegCom's primary phase of authentication in our calculus. The message-sequence chart below shows the steps in the case of $\mathcal{H} = \{\mathsf{V}_1\}$ and $Env = (\{RSU_1\}, \{\})$ and there is one node process RSU_1. In the analysis, we will consider more complex scenarios. We only show the trace of the vehicle V_1 in our calculus, as the trace of the RSU is less interesting. The nodes have default channel names in accordance with the calculus.



$$
\begin{aligned}
\mathsf{V}_1 := \quad & A_{single}(RSU_1).\\
& \overline{RSU_1}(Cert_{RSU_1}).\\
& \overline{RSU_1}\langle\{Cert_{V_1}, SK_{V_1,RSU_1},\\
& \qquad \{SK_{V_1,RSU_1}\}_{SK_{V_1}}\}_{ID_{RSU_1}}\rangle.\\
& D_{single}(RSU_1).\ 0
\end{aligned}
$$

4.2. UPPAAL Implementation

4.2.1. The Models

The models consist of one initiator, one responder, one attacker, and the environment. Figure 2 shows the UPPAAL model of an initiator. The initiator sends its certificate to the responder, and waits to receive $MV1$. In the UPPAAL model, the initiator signs the initial certificate. This is due to the fact that in literature certificates are typically authentic and contain information regarding who sent the certificate itself. Upon receiving $MV1$, the initiator decrypts it and decomposes it into its constituent parts, storing them locally only if they type check and the signature is successfully verified with the responder's public key. If this is all successful, the initiator then officially certifies the responder.

Figure 3 shows the UPPAAL model for the responder. The responder first receives a message and checks that this is a valid and authentic RSU certificate. The responder will have

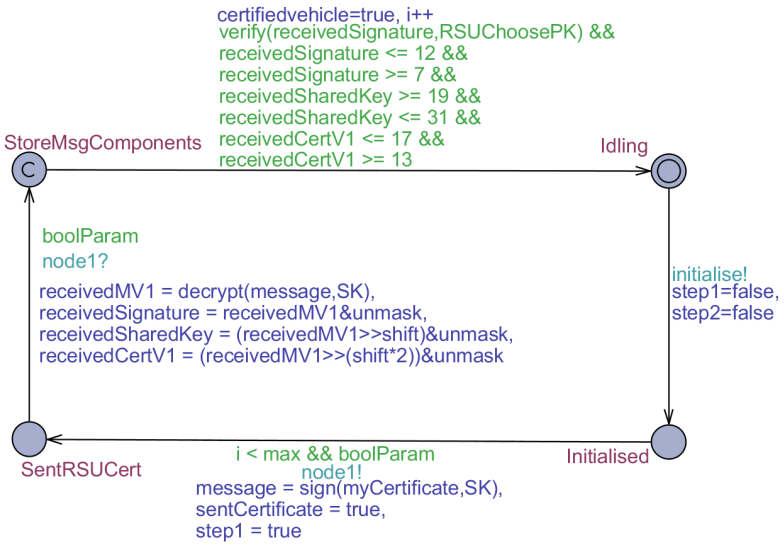


Figure 2: UPPAAL Model of the Initiator in the SegCom Protocol

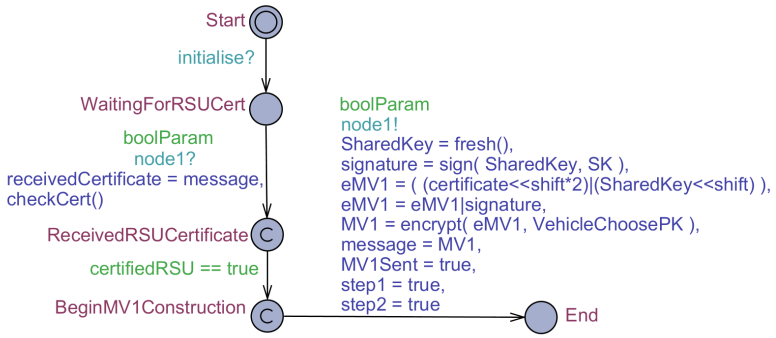


Figure 3: UPPAAL Model of the Responder in the SegCom Protocol

to move within range of an initiator in order to receive this message. If it is a valid certificate, it sets a global variable $certifiedRSU$ to true. The responder at this point is in the $BeginMV1Construction$ state. The subsequent transition builds the message $MV1$. The responder generates a fresh shared key and signs it with its secret key. The key and the signed key are both concatenated onto the end of the certificate sent by the initiator, and this is then encrypted with the ID of the initiator. The responder then sends the encrypted $MV1$ to the initiator on its default channel name.

Since message components will need to be concatenated and parsed, we define $shift$ to perform bit shifts on message components. Bits of one message component are shifted and a bitwise OR is applied on it and another message component to accomplish concatenation. In UPPAAL, this is accomplished with the expression $(messageComponent << shift)$ where $messageComponent$ is a message component, and a bitwise OR is accomplished with the |

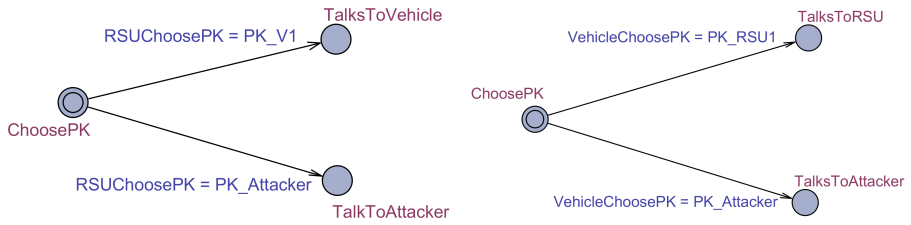


Figure 4: Choices of Keys for The Initiator and the Responder

operator. The variable $shift$ is fixed to be the length of each message component. Similarly, $unmask$ allows us to read each individual message component from a message.

To account for multiple scenarios that could take place with regards to who the agents choose to talk to (which public key they choose to encrypt messages), we provide separate scenario choice models. These choices are accounted for in Figure 4.

Figure 5 shows the attacker brain in the case of the SegCom protocol. This attacker brain is an implementation of the Dolev-Yao rules. The general mode of operation is as follows. The attacker reads in a message on any of the available channels, which are only available if the attacker agent has moved to a specific location (governed by $boolParam1$ and so on, similar to the initiator and responder). The received message is stored in a variable $data$. Depending on what step of the protocol the agents are in, the received message in $data$ can be analysed and stored in one of the variables $rsuCert$, $MV1$ or can be decomposed into $signature$, $SharedKey$, and $CertV1$. The attacker is now in the $DYAnalysis$ state at this point, and can choose to construct a new message from its existing knowledge, replay a message, or decrypt and analyse a received/eavesdropped message. If the attacker chooses the latter, it loads the message into variable $block$. A "block" is one or more message components that are concatenated together which are either about to be decrypted and analysed or about to be sent (or encrypted and sent). The attacker is now in the $ChooseKey$ state and can either choose to "stage" the block for sending, pick a key to encrypt the loaded block and stage this encrypted block for sending, or decrypt the block with its private key (other keys that they might know could be added as necessary). In a scenario where composed keys must be used, the attacker model will need to be adjusted to account for this, as the current model does not handle composed keys. Returning to the $DYAnalysis$ state, the attacker could choose to construct a message by transitioning to the $DYConstruction$ state. The attacker can load a message component from its knowledge into variable $data2$, to be concatenated with whatever is already stored in $data$. The variable i controls the maximum length of a message. The attacker can keep concatenating message components until $i == max\_op$, after which, whatever is stored in $data$ becomes a new block (as per restriction (2) Message Length) and the attacker returns to the $ChooseKey$ state.

Finally, we give the environment for this scenario in Figure 6. Note that when the vehicle moves out of the range of the RSU the key is no longer valid and the vehicle and RSU must carry out the protocol once more. This is justified in [7] by the authors mentioning that "the key is only valid for the segment (range of communication) for which it has been established." In order to capture movement among the agents, all the hubs (including the attacker agent) can be instantiated from the environment in order to achieve individual movement. The boolean

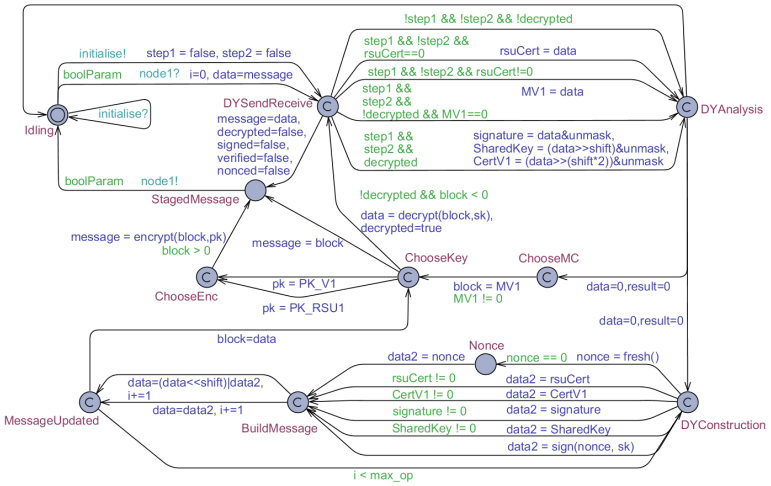


Figure 5: UPPAAL Model of the Attacker Brain in the SegCom Protocol

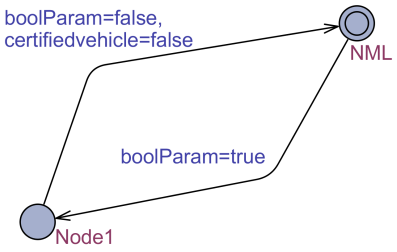


Figure 6: The Environment for the Scenario of the SegCom Protocol

parameters $boolParam1$, $boolParam2$, and $boolParam3$ are values that monitor the location of the hub, and the relevant boolean values for the instantiated hub are passed as parameters to the communication parts of the agents (i.e., initiator and responder) upon instantiation.

4.3. Instantiating the Scenarios

As an example of our approach in action, we focus on a simple environment where $Env = (\{RSU_1\}, \{\})$. RSU_1 has an accompanying node process RSU_1 and a single hub V_1 starts outside the communication range of RSU_1 and moves freely around the environment. Our scenario follows the typical execution of SegCom given in the message-sequence chart above, where we have instantiated a single vehicle which executes the protocol with RSU_1 only. The reason is that we are looking for attacks on a single execution of the protocol, but the environment is still instantiated so that the tool can also search for attacks relating to movement and location. The RSU is the initiator of the protocol and the vehicle acts as the responder. The attacker we instantiate moves as a hub, and has no prior knowledge other than public keys of all the other agents. The attacker is instantiated in no man's land similar to the honest vehicle.

By reference to the protocol in our calculus (see § 4.1), a maximum of three message com-

ponents can be concatenated together before encryption so we have chosen each message component to be 5 bits to avoid integer overflow. Ciphertexts are themselves message components, therefore an encryption of two message components is indeed one message component. There are four message / message component types to consider; the ciphertexts, the signatures, the certificates, and the *symmetric* keys. Since there are 5 bits per message component, we assign the following integer ranges to each type of message component: $1 \leq \text{Ciphertext} \leq 6$, $7 \leq \text{signatures} \leq 12$, and $13 \leq \text{certificates} \leq 18$, and $19 \leq \text{keys} \leq 31$. The ranges themselves are not special in any way, as long as there are enough integers left reserved for use by the protocol agents. For example, an integer range of 1 for the keys type is useless, as this means there is only one key that every agent would be forced to use. The value 0 also cannot be used, since we reserve 0 to represent $null$, or the absence of a message or message component.

4.4. Analysis

We use the UPPAAL verifier with its requirement language to specify security properties, and check that they hold for a given scenario. We report here on the analysis of relevant security properties. The SegCom protocol aims to establish mutual authentication between a vehicle and an RSU, so we analyse if security properties relating to authentication hold for the protocol.

4.4.1. Aliveness

In this case, we define the vehicle as being alive to the RSU if, when the RSU has certified the vehicle, it is true that the vehicle entered the communication range of the RSU and certified the RSU, which is formalised as follows:

Security Property 4.1 (Aliveness of Vehicle). $certifiedvehicle \dashrightarrow certifiedRSU$

The UPPAAL verifier finds Security Property 4.1 to hold, but it finds an attack in the scenario where the attacker has obtained the certificate of a vehicle via a side-channel (see the attack trace in Figure 7). This attack is unsurprising as the attacker knows the certificate of an honest vehicle already. The RSU simply chooses to talk to the attacker when they move within range, and the attacker can use this certificate in the $MV1$ message that they send to the RSU.

As an aliveness property of the RSU, we define that the RSU is alive to a vehicle if, when the vehicle sends message $MV1$, the RSU had previously sent its certificate:

Security Property 4.2 (Aliveness of RSU). $VehicleBrain.End \dashrightarrow RSU1.sentCertificate$

Security Property 4.2 is found to hold for this scenario. Unlike for Security Property 4.1, a separate scenario in which the attacker has been able to obtain the certificate of the RSU_1 via some side-channel attack does not yield an attack either. This is due to the fact that the certificates sent by RSU are signed.

4.4.2. Shared Key Agreement

The vehicle and RSU_1 should agree on the shared key that is established in a single protocol execution, immediately after $MV1$ is received, decrypted, and the signature verified by RSU_1.

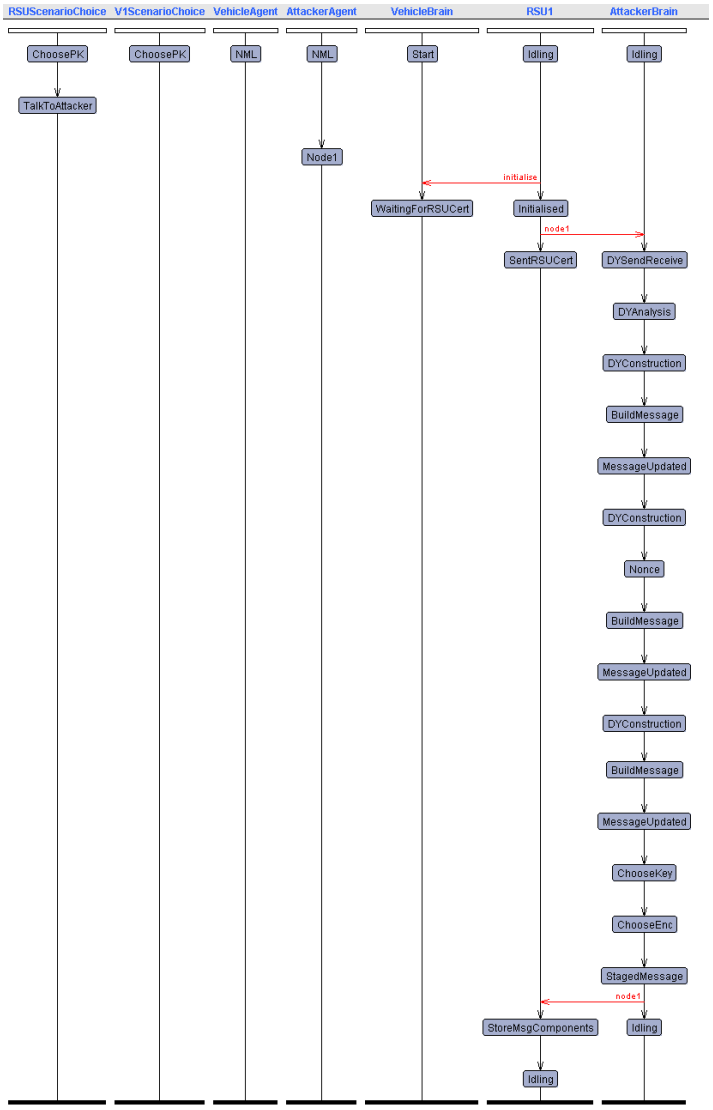


Figure 7: Trace of an Attack on Security Property 4.1

Security Property 4.3 (Shared Key Agreement). $certifiedvehicle$ *and* $RSU1.Idling$ *and* $certifiedRSU$ *and* $VehicleBrain.End$ $\dashrightarrow$ $VehicleBrain.SharedKey = RSU1.receivedSharedKey$

UPPAAL finds this strong key agreement to hold. If the initial certificate sent by the RSU is not signed, then UPPAAL finds the expected attack (the attacker impersonates the RSU).

4.4.3. Location Verification

Finally, if RSU_1 certifies the vehicle, then the vehicle must be situated in range of RSU_1.

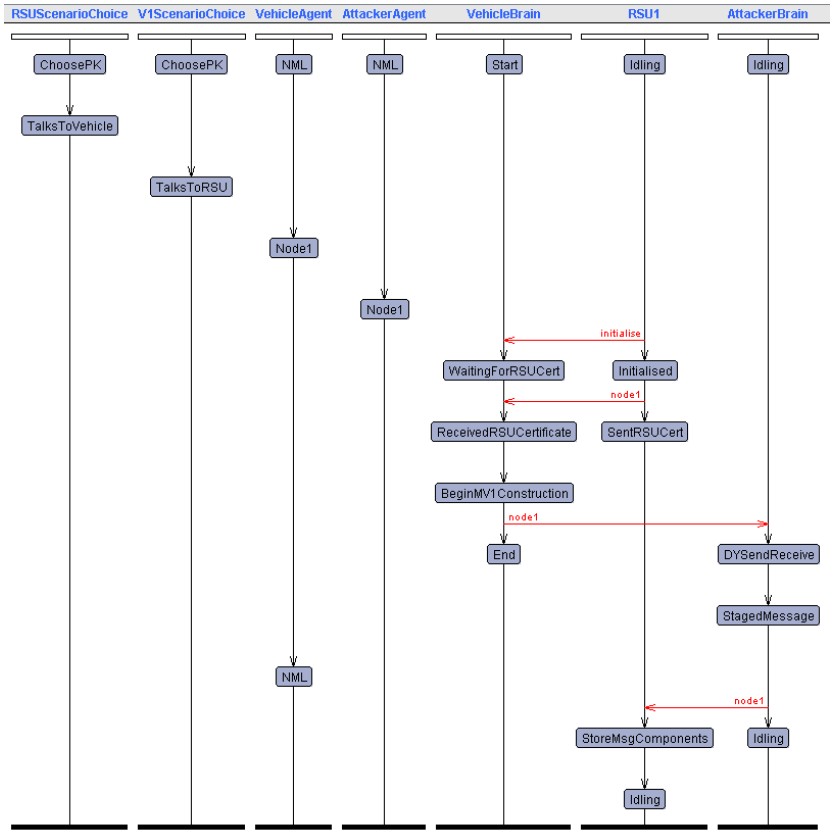


Figure 8: Trace of an Attack on Security Property 4.4

Security Property 4.4 (Location Verification). $certifiedvehicle \text{ and } certifiedRSU \dashrightarrow VehicleAgent.Node1$

For this property, UPPAAL finds an attack (Figure 8): the attacker intercepts the message $MV1$ and holds this message until the vehicle has left the communication range of the RSU. The attacker then finally sends the message $MV1$ to the RSU so that the RSU certifies a vehicle that is now far away. It is thus too late to have any subsequent communication with this vehicle.

5. Related Work

This is not the first time that location and movement have been considered in protocol analysis, so let us very briefly discuss how our approach is different from what has been done so far.

Mobile Ad-Hoc Networks (MANETs) that use cryptography are a hot research area (see, e.g., [12]) and some attacker models have been developed for MANETs (e.g., [13]), but, to the best of our knowledge, there does not exist a full-fledged, moving Dolev-Yao attacker for MANETs.

Distance-bounding protocols establish an upper bound on the physical distance between the participating agents [14], and are useful for proximity-dependent services such as contactless

payments or authorised entry to a space by scanning an ID card. Security protocol analysis approaches have been extended to consider, e.g., timestamps and the location of agents [15, 16, 17]. While there are some connections with this work, the attacker movement that we consider is quite different from that usually considered for distance-bounding protocols.

Applied pi-calculi such as [18] capture the movement of defined boundaries of computation known as *ambients* between administrative domains, and works like [19] added notions of security to express permission for an ambient to cross into another administrative domain. In our calculus, a location represents a physical finite range over which computation and communication can happen, and our notion of security comes from a protocol specification that exists between (mobile) agents that can communicate when they are in the same location.

Other works have used UPPAAL for security protocol analysis. In [10], Corin et al. carry out an analysis of protocols that involve timing aspects, exploiting the fact that UPPAAL allows clocks to be defined over models to monitor the passage of time as the system evolves. Our calculus of mobile agents models a notion of discrete time but we do not use clocks as we are not concerned with timing attacks. Corin et al. consider a Dolev-Yao-style intruder, but they do not include many message manipulation rules. They implement cryptography by way of a "cryptographic device", but we instead opt for the cryptography implementation in [11], where Koltuksuz et al. adopt an attacker model that is more powerful than that of [10] and use UPPAAL for the analysis of time-sensitive security properties on protocols, such as not receiving certain messages before they were sent and so on.

Our UPPAAL model of the attacker is based on the attacker "brain" featured in [11] combined with elements of [10], with some improvements and modifications of our own. The main novelty lies in the definition of an attacker model that consists of two parts: a novel attacker agent that formalises movement, and that interfaces with an attacker brain to send and receive cryptographic messages. For example, a message can only be received by the attacker brain if the attacker agent has moved to the required location. The second novelty is that the attacker can act as a agent of communication itself, rather than simply as a medium through which all communication passes. Our attacker can assume the role of a node or hub depending on the scenario considered, allowing for discovering new attacks either with movement or without. More modifications and improvements are described in § 3.3.

In this paper, we have shown that UPPAAL can be used to analyse protocols in the context of moving agents, and it comes as naturally as a timing-based analysis as in [10, 11]. However, UPPAAL is a generic tool in contrast to tools that have been designed specifically to analyse security protocols. For instance, many large-scale protocols have been analysed with Tamarin [5], e.g., [2] and [20] to name just a couple. Tamarin obviously fares much better at security protocol analysis than UPPAAL as it has been designed and further tailored to that purpose. However, to carry out the analysis that we want to with movement would involve extending the Tamarin tool itself or capturing movement semantics in a fairly non-trivial way. With UPPAAL, instead, capturing movement is much simpler and more flexible for our needs. This convenience is the reason why we chose to use UPPAAL, as a proof-of-concept, even though there is nothing in my approach that would prevent its use with Tamarin or one of the other more efficient security protocol analysis tools. In fact, we plan to do so in future work to analyse protocols with movement more complex than those considered here.

6. Conclusions and Future Work

Our calculus allows us to consider explicitly security protocols that require the agents to move to specific locations in order to exchange particular messages, along with a Dolev-Yao attacker who is able to move at will. With UPPAAL's support, we have automatically analysed some properties of the SegCom protocol, discovering attacks that rely explicitly on movement. These attacks result from our notion of an attacker which can be instantiated as a vehicle, whose movement and location are considered as a part of the analysis.

While this paper provides solid foundations, much work is left to be done, as have already mentioned in different places in the paper. In particular, we are working at extending the expressivity of our calculus to formalise movement in an even more granular way, and at improving our UPPAAL implementation to tackle its common termination issues. Since UPPAAL is convenient but not the main focus of our work, we will also consider using other tools. Still, we expect that we can improve UPPAAL's performance by extending the preliminary normalisation results given in [8]. We can make the attacker (and thus search) more effective by focussing only on attacks that are minimal, i.e., that consist of the fewest possible number of movements to carry out the attack with respect to a given environment. We are also working at automating the creation of environments and scenarios to widen the analysis capabilities.

References

[1] A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. Torabi Dashti, M. Turuani, L. Viganò, The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures, in: Proceedings of TACAS, LNCS 7214, Springer, 2012, pp. 267–282. doi:10.1007/978-3-642-28756-5_19.

[2] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, V. Stettler, A formal analysis of 5G authentication, in: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, 2018, pp. 1383–1396. doi:10.1145/3243734.3243846.

[3] C. J. F. Cremers, The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols, in: Proceedings CAV, LNCS 5123, Springer, 2008, pp. 414–418. doi:10.1007/978-3-540-70545-1_38.

[4] S. Escobar, C. Meadows, J. Meseguer, Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties, in: Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures, Springer, 2009, pp. 1–50. doi:10.1007/978-3-642-03829-7_1.

[5] S. Meier, B. Schmidt, C. Cremers, D. Basin, The TAMARIN prover for the symbolic analysis of security protocols, in: Proceedings of CAV, LNCS 8044, Springer, 2013, pp. 696–701. doi:10.1007/978-3-642-39799-8_48.

[6] D. Dolev, A. Yao, On the security of public key protocols, IEEE Transactions on information theory 29 (1983) 198–208. doi:10.1109/TIT.1983.1056650.

[7] M. Verma, D. Huang, SeGCom: secure group communication in VANETs, in: Proceedings

of the 6th IEEE Consumer Communications and Networking Conference, IEEE, 2009, pp. 1–5. doi:10.1109/CCNC.2009.4784943.

[8] A. Cook, L. Viganò, A Game Of Drones: Extending the Dolev-Yao Attacker Model With Movement, in: 6th Workshop on Hot Issues in Security Principles and Trust (HotSpot 2020). Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2020, pp. 280–292. doi:10.1109/EuroSPW51379.2020.00044.

[9] A. Cook, Automated Analysis of Security Protocols with Movement in UPPAAL, Ph.D. thesis, Department of Informatics, King's College London, London, UK, 2023. URL: https://archive.org/details/thesis_202305.

[10] R. Corin, S. Etalle, P. H. Hartel, A. Mader, Timed analysis of security protocols, Journal of Computer Security 15 (2007) 619–645. doi:10.3233/jcs-2007-15603.

[11] A. Koltuksuz, B. Kulahcioglu, M. Ozkan, Utilization of timed automata as a verification tool for security protocols, in: 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion, IEEE, 2010, pp. 86–93. doi:10.1109/SSIRI-C.2010.27.

[12] A. Hinds, M. Ngulube, S. Zhu, H. Al-Aqrabi, A Review of Routing Protocols for Mobile Ad-Hoc NETworks (MANET), International journal of information and education technology 3 (2013) 1. doi:10.7763/IJIET.2013.V3.223.

[13] J. Cordasco, S. Wetzel, An attacker model for MANET routing security, in: 2nd ACM conference on Wireless network security, ACM, 2009, pp. 87–94. doi:10.1145/1514274.1514288.

[14] S. Brands, D. Chaum, Distance-Bounding Protocols, in: Workshop on the Theory and Application of of Cryptographic Techniques, LNCS 765, Springer, 1993, pp. 344–359. doi:10.1007/3-540-48285-7_30.

[15] C. Cremers, K. B. Rasmussen, B. Schmidt, S. Capkun, Distance Hijacking Attacks on Distance Bounding Protocols, in: IEEE Symposium on Security and Privacy, IEEE, 2012, pp. 113–127. doi:10.1109/SP.2012.17.

[16] S. Mauw, Z. Smith, J. Toro-Pozo, R. Trujillo-Rasua, Distance-Bounding Protocols: Verification without Time and Location, in: IEEE Symposium on Security and Privacy, IEEE, 2018, pp. 549–566. doi:10.1109/SP.2018.00001.

[17] D. Singelee, B. Preneel, Location verification using secure distance bounding protocols, in: IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, IEEE, 2005. doi:10.1109/MAHSS.2005.1542879.

[18] L. Cardelli, A. D. Gordon, Mobile ambients, in: M. Nivat (Ed.), Foundations of Software Science and Computation Structures, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 140–155. doi:10.1007/BFb0053547.

[19] M. Bugliesi, G. Castagna, S. Crafa, Reasoning about security in mobile ambients, in: Proceedings of CONCUR, LNCS 2154, Springer, 2001, pp. 102–120. doi:10.1007/3-540-44685-0_8.

[20] T. Chothia, M. D. Ryan, Modelling of 802.11 4-Way Handshake Attacks and Analysis of Security Properties, in: STM 2020: Security and Trust Management, LNCS 12386, Springer, 2020, pp. 3–22. doi:10.1007/978-3-030-59817-4_1.