Verifying Security Properties in Unbounded Multiagent Systems

Ioana Boureanu
Department of Computing
Imperial College London, UK
i.boureanu@ic.ac.uk

Panagiotis Kouvaros
Department of Computing
Imperial College London, UK
p.kouvaros@ic.ac.uk

Alessio Lomuscio
Department of Computing
Imperial College London, UK
a.lomuscio@ic.ac.uk

ABSTRACT

We study the problem of analysing the security for an unbounded number of concurrent sessions of a cryptographic protocol. Our formal model accounts for an arbitrary number of agents involved in a protocol-exchange which is subverted by a Dolev-Yao attacker. We define the parameterised model checking problem with respect to security requirements expressed in temporal-epistemic logics. We formulate sufficient conditions for solving this problem, by analysing several finite models of the system. We primarily explore authentication and key-establishment as part of a larger class of protocols and security requirements amenable to our methodology. We introduce a tool implementing the technique, and we validate it by verifying the NSPK and ASRPC protocols.

1. INTRODUCTION

Formalisms grounded in the multi-agent systems (MAS) paradigm have made a significant contribution to the development of a wide range of applications, including search and rescue [19], automatic negotiation [13], and security [9]. For the MAS paradigm to continue to drive the sound development of forthcoming topical applications, the underlying MAS formalisms need to evolve to capture and solve the theoretical challenges that arise in these applications.

The "internet of things" (IoT) is an important area of current and future growth, where MAS can play a leading role. In current IoT applications networked objects equipped with sensors and computing capabilities exchange data over the Internet. It is expected that some of these objects will become autonomous and will independently cooperate and negotiate with their peers while representing the needs of a human user or an organisation. Security and privacy aspects of IoT applications remain a concern. Users may not be willing to adopt the technology if the data being exchanged can be directly traced back to them, or tampered with by unauthorised third parties. It is therefore important to guarantee that the protocols run by IoT applications are robust and cannot be hijacked by attackers. One important property is the correctness of the underlying authentication and key-establishment procedures.

Of course these have long been the object of research in security. Several noteworthy protocols, including versions of secure RPC [30] discussed herein, have been analysed by a variety of methods, including model checking [5]. However, these results cannot in general be applied to IoT. This is because verification of security protocols traditionally analyses a bounded number of interacting principals. In contrast, IoT's unbounded connectivity intrinsically calls for the need to verify an arbitrary number of protocol executions.

The aim of this paper is to put forward a novel methodology for the unbounded verification of a class of security protocols. A key feature of our work is that, similarly to [7], we also capture the epistemic properties of the principals in a secure exchange. To do so we develop a Dolev-Yao threat model in a MAS setting (Section 2); we define and solve the parameterised model checking problem for these systems (Section 3); we present a tailored verification toolkit for the automatic verification of protocols described in a new, security-oriented language (Section 4); and we showcase the methodology on NSPK [27] and ASRPC [30]. In doing so, we answer the open question of model-checking an unbounded number of protocol executions against temporal-epistemic properties.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

2. PARAMETERISED SECURITY MODEL

High-level protocol descriptions.

Consider the following high-level description of the well-known Neeham-Shroeder public-key (NSPK) protocol [27]. At step 1, A encrypts the concatenation of her identity and a nonce n_A with the public key of B and sends this to B. At step 2, upon receipt and decryption, B concatenates a new nonce n_B to the received n_A, encrypts this with the public key of A and sends the overall result to A. The rest follows similarly. We refer to the full representation of A's variables and actions as the *A-role*.

`The High-Level Description of NSPK`

`Protocol:`
1. $A \rightarrow B:\ enc((A, n_A); pub_B)$
2. $B \rightarrow A:\ enc((n_A, n_B); pub_A)$
3. $A \rightarrow B:\ enc(n_B; pub_B)$

`Longterm Vars:` $priv_A; pub_A; A; priv_B; pub_B; B;$
`Prot. Vars:` $n_A, n_B : nonce;$
n_A `bound` A; n_B `bound` B
`Goals: A:B:auth`(A, n_A, pub_A); `secret`(n_A);

In protocol executions, high-level roles are embodied by *parties* with concrete credentials. If a party acts according to

an A-role, we call it an *A-party*. `Longterm Vars` refer to data belonging to parties and not varying with protocol executions: an A-party called *alice* has $skey_{alice}$ as her long-term secret key. In turn, the `Prot. Vars` part contains sub-messages which are freshly generated in each new, correct session: e.g., n_A is a nonce. It originates from an A-party, as "n_A `bound` A" stipulates. So, we say that n_A is *bound* to the A-role. Contrarily, n_B is *free* in the A-role, as n_B is *bound* to the B-role. `NSPK`'s `goals` here amount to two security requirements. By "`A:B:auth`(A, n_A, pub_A)" it is meant that in any session, the B-party is convinced of communicating to a specific A-party on the basis of A, n_A, and pub_A, and that it was this party who sent the data. By "`secret`(n_A)", it is intended that in any one session, n_A should only become known to the two participating A- and B-parties.

Intermediate level protocol description.

Any protocol Pr can be described algebraically by considering: 1) a set S of sorts (e.g., $Keys$, $Nonces$); 2) a sort-indexed set of variables $X = (X_s \mid s \in S)$ – in `NSPK`, for $s{=}Nonces$, $X_s = \{n_A, n_B\}$; 3) a signature Σ over X, which then gives a set $T_{\Sigma,X}$ of terms to symbolise the protocol-messages, e.g., $enc((A, n_A); pub_B)$ for `NSPK`; 4) a universal algebra $\mathbb{A}$ with *infinitely countable* support-sets A_s, used to interpret Σ; 5) a term algebra $\mathbb{T}$; 6) a set or theory E of equations over terms encoding the cryptographic operations on them.

Assignments δ of variables into the algebra $\mathbb{A}$ and their homomorphic extension to terms, actions and roles model protocol executions. At the start, parties ignore the values of variables free in their role (like n_B for `NSPK`'s A-role). To denote this, we use *null values* denoted $\perp$ or $\perp_s$ for a specific sort $s \in S$. We extend $\mathbb{A}$ to the algebra $\mathbb{A}_\perp$, which has $A_{s_\perp}{=}A_s \cup \{\perp_s\}$ as support-sets and all operations over null values returning null values.

Shorthands & building blocks.

We call a protocol *receiver-transparent* if receivers can decipher messages down to variables upon their receipt. `NSPK` and many other authentication and key-agreement protocols [31] are receiver-transparent.

As in protocol-descriptions, we differentiate between longterm variables ($\mathcal{LT}$), free variables ($\mathcal{F}_R$) and bound variables ($\mathcal{B}_R$) in some role R. This extends naturally to terms (i.e., if a term contains a free variable it is free in that role, otherwise it is bound to that role). On this, we give the range of a term homomorphically, as follows.

Definition 2.1 (Range of a term for role). *The* range $Range_R(t)$ of a term $t \in T_{\Sigma,X}$ for a role R *is as follows:*

$$Range_R(t) = \begin{cases} A_s & \text{if } t \in (\mathcal{B}_R)_s \cap X_s \\ A_{s_\perp} & \text{if } t \in (\mathcal{F}_R)_s \cap X_s \\ (A_{s_1} \times \ldots \times A_{s_n}) \cup A_s & \text{if } t \in (\mathcal{B}_R)_s, t = \sigma(t_1, \ldots, t_n), \\ & \sigma \in \Sigma_{(s_1,\ldots,s_n),s}, t_i \in T_{\Sigma,X,s_i} \\ (A_{s_1} \times \ldots \times A_{s_n}) \cup A_{s_\perp} & \text{if } t \in (\mathcal{F}_R)_s, t = \sigma(t_1, \ldots, t_n), \\ & \sigma \in \Sigma_{(s_1,\ldots,s_n),s}, t_i \in T_{\Sigma,X,s_i} \end{cases}$$

The *maximal range of a term* t, written $Range(t)$, is defined as the union of $Range_R(t)$ over all the roles R. Ranges of long-term variables do not vary with roles: for $t \in \mathcal{LT}$, $Range(t) = A_\omega$, where $\omega \in S^*$ is the type of t. We consider a variable called *step* of sort *Steps*, to encode the actual protocol steps. Ranges $Range_R(step)$ of steps for a given role R are finite and in line with the protocol description.

Let $Msg \subsetneq T_{\Sigma,X}$ denote strictly the protocol-messages. For a protocol-role R, let the sets $SentMsg_R \subsetneq Msg$ and $RecvMsg_R \subsetneq Msg$ respectively denote the messages sent and received by R. We introduce a functional symbol *out* of type $(SentMsg_R; Steps)$, which is interpreted as the particular protocol-step at which a message is sent. We write $out(t) = x$ to denote that a message t is sent at step x, i.e., in `NSPK`, $out(enc((A, n_A); pub_B))$=1. Similarly, let *in* be a functional symbol of type $(RecvMsg_R; Steps)$ which is interpreted as the protocol-step at which a concrete message is to be received.

Let $Sub(t)$ be the set of proper sub-terms of t and $Vars(t)$ denote the variables in t. In `NSPK`, $Vars(enc((A, n_A); pub_B))$ $=\{A, n_A, pub_B\}$ and $(A, n_A) \in Sub(enc((A, n_A); pub_B))$.

Templates and parameterised systems for security.

Interleaved interpreted system (IIS) [23] are a multi-agent system semantics in which agents synchronise on one sole action to be performed at each point in time. IIS components can be described in a parametric way via *templates* [21]. The *instances* of these templates, also called *agents*, unravel an actual, *concrete system*. Next, we will present a security model based on templates.

Definition 2.2 (Template (adapted from [21])). *A* template *is a tuple $T{=}(X, L, \iota, Act, P, t)$, where X is a set of variables, $L \subseteq X \times A$ is a set of local states describing variables X with their values in the support sets of the algebra $\mathbb{A}$, $\iota \in L$ is a unique initial local state, Act is a nonempty set of actions defined over X, $P : L \to Act$ is a protocol function dictating which action can be performed at a local state, tr: $L \times Act \to L$ is a local evolution function specifying state-updates caused by actions. A "null" action $\lambda \in Act$ can be performed at any state and yields no state update.*

Templates here differ from [21] in that they have an algebraic nature, with sorted variables and ranges over infinitely countable sets of a term-algebra. We will also partition these sets driven by a security semantics. This will then open for new techniques of instantiating templates into agents. For instance, agents herein will be able to share variables and their ranges.

We say that an action is *executable* in a template if given a concrete, local state of some arbitrary instance of this template, this action can take place at that local state.

We say that an executable action is *setting* in a template if once this action is performed at a local state of some arbitrary instance of this template, then that local state is consequently updated.

Protocol roles to templates.

Let R be a protocol-role in a receiver-transparent protocol Pr. We will map each such protocol-role R into a *template for* R, written T-R.

T-R's variables. These are described by the set $X_{T\text{-}R}{=}\{\mathcal{LT} \cup \mathcal{F}_R \cup \mathcal{B}_R\}$.

So, T-R's variables distinguish between those free and those bound in R, as well as longterm variables.

The local states of T-R. These are described by the relation $L_{T\text{-}R}{=}(t{::}Range_R(t) \mid t \in X_{T\text{-}R})$ between terms and their respective ranges for the R-role.

The local states for the T-A in `NSPK` are as follows: $(step{::}\mathbb{N}, n_A{::}\mathbb{N}, n_B{::}\mathbb{N}_\perp, pub_B{::}Keys_\perp, A{::}ID)$, where $step, n_A \in \mathcal{B}_A$, $n_B, B \in \mathcal{F}_A$ and $A \in \mathcal{LT}$.

In the *initial state ι of the T-R*, we assign bound and long-term variables to arbitrary non-null values over their ranges, free variables to null, and all steps are set to 1.

Actions for templates for roles. The actions of the T-R template are given by:
$Act_{T\text{-}R} = \{send\_intercept(t_1), receive\_transmit(t_2), \lambda\}$, for all $t_1 \in SentMsg_R$ and $t_2 \in RecvMsg_R$.

So, templates's actions encode the sending and receiving of abstract messages, or perform the empty action λ. For the operational semantics, we need a 'handle' to concrete values and template-instances. This is as follows.

Representative values in T-R. Let $t \in T_{\Sigma,X}$. A *representative value for t in T-R*, written $val_{T\text{-}R}(t)$ or simply $val(t)$, is the value in $Range_R(t)$ held for t in an arbitrary instance of T-R at a given point.

We now give the operational semantics of T-R'a actions.

Operational sending actions. Let $t \in SentMsg_R$. An *action $send\_intercept(t)$ is executable in T-R* if $val(v) \neq \bot$, for all $v \in Vars(t)$.

In other words, an instance of T-A in `NSPK` needs to have non-null values for n_B, i.e., $val(n_B) \neq \bot$, in order to send the `NPSK`'s third message m_3. Its initial state appears as $(step = 1, n_A = n, n_B = \bot, \ldots)$; the $send\_intercept(m_3)$ action is enabled only once it is in a state $(step = 1, n_A = n, n_B = n' \ldots)$, with $n, n' \in \mathbb{N}$.

Operational receiving actions. Let $t \in RecvMsg_R$. An *action $receive\_transmit(t)$ is executable in T-R* if the following holds: for each $v \in Vars(t)$, if $val(v) \neq \bot$ then $val(v) = \delta(t)|_v$, where $\delta(t)$ is a concrete value for t now being sent to an instance of T-R and $\delta(t)|_v$ returns the value of v within $\delta(t)$.

An executable action $receive\_transmit(t)$ is *setting in T-R* if for all $v \in Vars(t)$ such that $val(v)=\bot$, the application of $receive\_transmit(t)$ changes $val(v)$ to $\delta(t)|_v$.

Thus, a receive-transmit action is executable if any value received at some point is consistent with data held previously and locally. This is in line with the matching receiving semantics in [29]. In `NSPK`, when an A-party receives a value $\delta(enc((n_A, n_B); pub_A))$ for the second message, she checks that her previously used value $val(n_A)$ for n_A matches the n_A-value $\delta(enc((n_A, n_B); pub_A))|_{n_A}$ sent via $\delta(enc((n_A, n_B); pub_A))$. An executable receive-transmit action is *setting* if it causes variable-updates in concrete instances of templates. In `NSPK`, a local value for n_B would be extracted out of the received $\delta(enc((n_A, n_B); pub_A))$ and $val(n_B)$ would be consequently updated.

Local protocol for T-R. Let l be a local state of T-R, $t_1 \in SentMsg_R$, $t_2 \in RecvMsg_R$. Then, the local protocol function $P_R : L_{T\text{-}R} \rightarrow Act_R$ is defined as follows:

$$\begin{cases} P_R(l) = send\_intercept(t_1), & \text{if } v(step) = out(t_1), \\ & send\_intercept(t_1) \\ & \text{is executable in } T\text{-}R; \\ P_R(l) = receive\_transmit(t_2), & \text{if } v(step) = in(t_2), \\ & receive\_transmit(t_2) \\ & \text{is executable in } T\text{-}R. \end{cases}$$

If an action a is such that $a \in P_R(l)$, we say that *a is enabled (at l) in T-R.*

So, any T-R instance is able to perform an action if she is at the right protocol step for it and if the action is executable (i.e., all values to send are non-null and values to receive are consistent with those already held).

Local evolution for T-R. Let $a_1, a_2 \in Act_R$ such that $a_1 = send\_intercept(t_1)$ and $a_2 = receive\_transmit(t_2)$. Let l be an arbitrary local state of T-R and $val(\cdot)$ denote values held by some arbitrary instance of T-R at their instance of l.

The local evolution function is tr_R: $L_{T\text{-}R} \times Act_R \rightarrow L_{T\text{-}R}$ with $tr_R(l, a_i) = l'$ if $a_i \in P_R(l)$ (for $i \in \{1, 2\}$) and the following holding:

$$\begin{cases} val(step) + +, & \text{if } a_i = a_1 \quad (1) \\ val(step) + + \text{ and } a_2 \text{ is setting}, & \text{if } a_i = a_2 \quad (2), \end{cases}$$

where $val(\cdot)$ denotes values held at l'.

The above says that enabled actions, once performed, update the current local state with incremented protocol-steps, and receiving actions may set new values.

The Environment template.

We will now present a special template called the *Environment template* and written T-Env. It models the outer network hosting both the honest parties' from the adversary. In a concrete multi-agent system, T-Env is instantiated into *concrete Environment* which we denote by Env.

Local states for T-Env. These are defined by a series of variables, emulating the following map for each role R: $log[t, \{v :: val \mid v \in Vars(t)\}]$ with $t \in SentMsg_R$, and $val \in Range(v) \cup \{\bot\}$.

That is, T-Env tracks the values of the variables v inside each protocol-message t. For $enc((A, n_A); pub_B)$ in `NSPK`, T-Env stores three variables for A, n_A, and pub_B.

In *T-Env's initial states* all the log are set to null-values: i.e., $log[t][v] := \bot$, for all $v \in Vars(t)$, for all $t \in Msg$, denoting that no tracking has yet occurred.

The concrete environment Env contains in fact several copies of a $log[t, \cdot]$ map, one reserved for each instance of T-R having emitted a value for t, as depicted below.

Representative values in T-Env. Let $t \in SentMsg_R$. *Representative values for t in T-Env* are the set of arbitrary values in $2^{Range(t) \cup \bot}$ held in Env for $Vars(t)$ under the handle t^{rep} of t, written $log[t^{rep}, \{v \mid v \in Vars(t)\}]$.

So, if an instance rep of T-A sends `NSPK`'s m_1 as some $\delta(enc((A, n_A); pub_B))$, then Env sets $log[m_1^{rep}][n_a]$ to $\delta(m_1)|_{n_a}$, sets $log[m_1^{rep}][A]$ to $\delta(m_1)|_A$, etc., where $log[\cdot^{rep}][\cdot]$ is a rep-reserved' copy of $log[\cdot][\cdot]$.

Actions for T-Env. These are given by the set $Act_{T\text{-}Env} = Act_{T\text{-}R} \cup \{tamper\_log(t)\}$, for all R-roles, for all $t \in RecvMsg_R$.

The $tamper\_log(t)$ actions refer to the threat model; we will describe these later in the section. The other actions are in tandem with T-R: if T-R sends something, then T-Env intercepts it, and if T-R receives something it is because T-Env transmits it.

Let $t \in SentMsg_R$, $t' \in RecvMsg_R$ be as above. Let rep be an arbitrary instance of T-R at a given point.

Operational intercept. The *action $send\_intercept(t)$ is executable in T-Env* at any local state of T-Env.

An *action $send\_intercept(t)$ is setting in T-Env* in that $log[t^{rep}][v]$ becomes $\delta(t)|_v$, for all $v \in Vars(t)$, where $\delta(t)$ is the arbitrary concrete value for t sent within rep and $\delta(t)|_v$ returns the value of v within $\delta(t)$.

The second part of the above says that all send-intercept actions are setting: they make the corresponding logs of Env's take the respective values from the messages sent.

Operational transmit. The *action* $receive\_transmit(t')$ *is executable in* T-Env if $log[t'][v] \neq \bot$, for all $v \in Vars(t')$.

A receive-transmit action is executable in T-Env if all the necessary sub-parts of the message to be received by a T-R instance have been logged by T-Env, symbolising that the concrete message is present over the network.

The *local protocol or local evolution function for* T-Env are implicitly specified by the operational semantics above.

The Intruder Template.

We will now present a special template called the *Intruder template*, written T-In, which encapsulates a Dolev-Yao (DY) threat model [14]. We instantiate the T-In template by one agent only which we call $Intr$ or *concrete intruder*.

T-In's local states. These are defined by the following:
1) a set of variables, emulating the maps $constr[t, \{v :: val \mid v \in Vars(t)\}]$, for each role R, with $t \in SentMsg_R$, and $val \in Range(v) \cup \{\bot\}$;
2) a series of boolean variables $poss(val)$, for each $val \in Range(v)\backslash\{\bot\}$, for each $v \in Vars(t)$, for each $t \in Msg$;

The T-In's $constr$-variables capture the protocol's messages as formed by the attacker. The $poss$-variables denote the intruder's data-possessions. I.e., $poss(alice) = .T.$ denotes that the intruder knows the name *alice* to be a protocol participant; $poss(300) = .F.$ encodes that the intruder cannot use 300 as a fresh value.

Representatives for T-In. Let rep be an arbitrary instance of T-R. All term t linked to rep that the intruder tracks or manipulates w.r.t. to this instance is referred to as t^{rep}.

Like with Env, the concrete $Intr$ would replicate $constr[t, \cdot]$ into $constr[t^{rep}, \{v :: val \mid v \in Vars(t)\}]$ for each rep.

In T-In*'s initial states*, all the $constr$-variable are set to null-values; some of the $poss$-variables are set to $true$, some are set to $false$ denoting the intruder initial knowledge. This is done in such a way that T-Rs and T-In share no value for variables that are not long-term.

Actions for T-In. The actions of the T-In template are given by: $Act_{Intr} = \{DY(t_1), tamper\_log(t_2), \lambda\}$, for all R-roles, for all $t_1 \in SentMsg_R$, for all $t_2 \in RecvMsg_R$.

We introduce a shortcut to intercepts by the intruder: the $Intr$ is allowed conceptual access to the values of messages inside any arbitrary instance of T-R. He performs closure of DY-analysis and DY-synthesis [28] of these messages. This is formalised below.

Operational DY actions. Let rep be an arbitrary instance of T-R, $t \in SentMsg_R$, $val(t)$ be the homomorphically composed value of t within rep, $t_1 \in RecvMsg_{R'}$, and rep_2 be an arbitrary instance of T-R'.

An *action* $DY(t)$ *is executable in* T-In if $constr[t^{rep}][v] = \bot$, for all $v \in Vars(t)$.

An executable $DY(t)$ action *is setting in* T-In if the following is the case:

$$\begin{cases} poss[x] = .T., & \text{if } x \in analz(val(t), poss),\ x \neq \bot \quad (1) \\ constr[t_1^{rep_2}][v] := y, & \text{if } y \in \overline{synth(analz(poss), t_1)},\ y \neq \bot \quad (2) \end{cases}$$

By point (1) above, once a DY action is performed, it sets values in $poss$ upon analysis of the value of t inside T-R's instance rep. As per (2) above, the values in $poss$ are then used to try and compose a value for message t_1: i.e., $\overline{synth(analz(poss), t_1)}$ denotes closure of DY synthesis and analysis over all $Intr$ possessions, for the construction of t_1. If successful, the synthesised value y is recorded in that $constr[t_1^{rep_2}][v] := y$.

We took $DY(t)$-action to be executable if $constr[t^{rep}][v]$ is $\bot$; so, we see now that this equates to $Intr$ decomposing values of messages t that he has not formed himself. For example, let instance rep of T-A for `NSPK` have a local value $\delta(m_1)$ for the first message $enc((A, n_A); pub_B)$ such that for the value $\delta(m_1)|_{pub_B}$, $Intr$'s $poss$ has $\delta(m_1)|_{priv_B}$ set to true. Then $Intr$ can decrypt $\delta(m_1)$ and, according to (1) above, $poss(v_1)$ becomes $.T.$, where $v_1 = \delta(m_1)|_{n_a}$. Next, according to (2), the DY action is such that $constr[t'^{rep_2}][n_a] := v_1$, where rep_2 is another instance of T-A and $Intr$ is now constructing a message replaying in it the value of n_a as per rep_1 possibly purporting to be from rep_2.

The operational semantics for the $tamper\_log$-actions in T-In and in T-Env follows.

Operational tampering actions. Let $t \in SentMsg_R$.
An *action* $tamper\_log(t)$ *is executable in* T-In if $constr[t][v] \neq \bot$, for all $v \in Vars(t)$.

An *action* $tamper\_log(t)$ *is executable in* T-Env at any local state of T-Env.

Let rep be an arbitrary instance of T-R. An executable *action* $tamper\_log(t)$ *is setting in* T-Env in that $log[t^{rep}][v]$ becomes $constr[t^{rep}][v]$ in Env, for all $v \in Vars(t)$.

The operational semantics of tampering actions stipulates that $Intr$ is able to perform a $tamper\_log(t)$-action if it has constructed all the parts needed in t, i.e., it holds a value $v \neq \bot$, for all the variables of t. In T-Env, these actions are executable at any point and have as effect the injection of the intruder's constructions in $constr$ in Env's the log-vars.

The *local protocol or local evolution function for* T-In are specified implicitly via the operational semantics above.

Definition 2.3 (Parameterised security system). *A* parameterised security system (PSS) *for a protocol* Pr *is a tuple* S=$(\mathcal{T}, T$-$Intr, T$-$Env)$, *where* $\mathcal{T} = \bigcup_{R-role\ in\ Pr}\{T$-$R\}$, *with the components defined as above.*

3. UNBOUNDED SECURITY IN MAS

A *concrete instance of template* T-R is defined constructively and inductively based on the instances who already 'joined' a concrete system modelled by a PSS. A non-basic step of this inductive definition is given below. For the next two definitions, we assume w.l.o.g. that there are two roles R and R' in the protocol Pr.

Concrete agents.

Definition 3.1 (Concrete role-agents). *Let* R *be an arbitrarily fixed role and* T-$R = (X = \mathcal{B}_R \cup \mathcal{F}_R \cup \mathcal{LT}, L \subseteq X \times A, \iota, Act, P, tr)$ *as above. Let* $m \geqslant 1$ *be the total number of instances present in the system. Let* $n \geqslant 1$ *be the total number of instances of* T-R *present in the system. Let* i *with* $1 \leqslant i \leqslant n$. *The* (i, n, m)-concrete instance of T-R, *written* ag_R^i *or for simplicity* ag, *is defined by the* $(X^{ag}, L^{ag}, \iota^{ag}, Act^{ag}, P^{ag}, tr^{ag})$ *tuple as follows:*
I. X^{ag} *is a multi-copy of* X, *where:*

1). If $x \in \mathcal{B}_R$, then $x^{ag} \in X^{ag}$;
2). If $y \in \mathcal{F}_R \cap \mathcal{B}_{R'}$, then $y_1^{ag}, \ldots, y_{m-n}^{ag} \in X^{ag}$;
3). If $z \in \mathcal{LT}$, then $z_1^{ag}, \ldots, z_m^{ag} \in X^{ag}$.
II. $L^{ag} \subseteq X^{ag} \times 2^{Range(X)}$ is as follows:
1). Let $x \in \mathcal{B}_R \cup \mathcal{LT}$. Then, $(x, Range_{ag}(x)) \in L^{ag}$ where $Range_{ag}(x)$ is a subset of $Range_R(x)$ with $|Range_{ag}(x)|=n+1$ such that $Range_{ag'}(x) \supsetneq Range_{ag}(x)$, where ag' is the (i, n', m')-concrete instance of T-R and $n' > n$;
2). Let $x \in X_{Steps}$. Then, $(x, Range_R(x)) \in L^{ag}$.
III. Act^{ag} is a multi-copy of Act, as local terms are expanded over X^{ag};
• *Any $a \in Act$ is send_intercept(t) or receive_transmit(t), with $t \in Msg$. Then, $a_1^{ag}, \ldots, a_s^{ag} \in Act^{ag}$, where $s = |Range_{ag}(t)|$ with $Range_{ag}(t) = \times_{x \in Vars(t)} Range_{ag}(x)$.*
IV. ι^{ag}, P^{ag} and tr^{ag} are defined as on T-R expanded over X^{ag}, L^{ag}, Act^{ag} above and homomorphically instantiated.

Point I.1 in Definition 3.1 says that the agent keeps just one single local copy of each variables bound to its role. By I.2, for each party that may provide ag with a value for a variable free in its role, ag holds a separate copy of that variable. Indeed, an A-party in NSPK may receive n_b's for all the B-parties that she engages with. Finally, I.3 describes the fact that long-term values may be associated to all m parties in the system, i.e., the agent has public-keys of all the parties she communicates to. Point II shows that we capture a truly unbounded system and we expand the ranges of variables on-the-fly, as more instances 'join' the system. Point II.1 expresses that constructing a new instance of T-R makes the ranges of its bound variables grow: at the i-th instance, this range has a size of $n+1$ (as n could cause logic omniscience). Point II.2 stipulates that protocol-steps do not vary with instances. And, upon point III, an agent will have send or receive actions over all the possible values (in $Range_{ag}(t)$) derived homomorphicaly from the variables x in t and their ranges defined as per point II.

Definition 3.2 (Concrete (n,m)-intruder). *Let T-$Intr = (X, L \subseteq X \times A, \iota, Act, P, tr)$ as above. Let $m \geqslant 1$ be the number of all role-agents present in the system. For an arbitrary role R, let all $n \geqslant 1$ instances of T-R therein define the set Ag. The* concrete (n,m)-intruder *$Intr(n,m)$, or simply $Intr$, follows the definition of T-In, where:*
I. X^{Intr} is a multi-copy of X expanded by Ag:
1). For each $t \in SentMsg_R$, for each $v \in Vars(t)$, for each ag, we have that $x \in X^{Intr}$, where $x = constr[t^{ag}][v]$;
Then, let $Range_{Intr}(x)$ be $\bigcup_{ag \in Ag} Range_{ag}(v) \cup \mathcal{R}_1 \cup \mathcal{R}_2$, where $\mathcal{R}_1 \subseteq \{Range(v) \backslash \{\bigcup_{ag \in Ag} Range_{ag}(v)\} | v \notin \mathcal{LT}\}$ and $|\mathcal{R}_1| = n$, $\mathcal{R}_2 \subseteq \{Range(v) \backslash \{\bigcup_{ag \in Ag} Range_{ag}(v)\} | v \in \mathcal{LT}\}$ and $|\mathcal{R}_2| = m$.
2). Let $x' \in X^{Intr}, x' = constr[t][v]$ as above. For each val in $Range_{Intr}(x')$, let $x = poss[val]$. Then, $x \in X^{Intr}$; Also, let $Range_{Intr}(x)$ be $\{true, false\}$.
II. $L^{Intr} \subseteq X^{Intr} \times 2^{Range(X)}$ where $(x, Range_{Intr}(x)) \in L^{ag}$ for each $x \in X^{Intr}$ as above.
III. Act^{Intr} is a multi-copy of Act, as local terms and their values are homomorphically expanded over $x \in X^{Intr}$ and $Range_{Intr}(x)$. The resulting $DY(\cdot)$ actions are denoted Act–DY^{Intr}.
IV. ι^{Intr}, P^{Intr} and tr^{Intr} are as in T-$Intr$ expanded over X^{Intr}, L^{Intr}, Act^{Intr} and homomorphically instantiated.

Point I.1 in Definition 3.2 shows that the intruder can compose and decompose messages over all the respective values inside the role-agents present in the system. Then, by the set $\mathcal{R}_1$, we see that for each role-agent that "joins" the system, $Intr$ gets a new possible value for each variable that is not long-term; so, he could inject all needed fresh values in each session. The set $\mathcal{R}_2$ shows that he can get long-term values for all instances; so, he could potentially be a party in any session using appropriate long-term keys. The rest is self-explained or explained in the template T-$Intr$.

Definition 3.3 (Concrete (n,m)-environment). *Let T-$Env = (X, L \subseteq X \times A, \iota, Act, P, tr)$ as above. Let $m \geqslant 1$ be the number all role-agents present in the system. For an arbitrary role R, let all $n \geqslant 1$ instances of T-R therein define the set Ag. Then, the concrete (n,m)-environment $Env(n,m)$ follows the definition of T-Env such that:*
• *X^{Env} is a multi-copy of X, where for each $t \in SentMsg_R$, for each $v \in Vars(t)$, for each ag instance of T-R, we have that $x \in X^{Env}$, with $x = log[t^{ag}][v]$;*
Let $Range_{Env}(x)$ be $Range_{Intr}(x)$;
• *$L^{Env} \subseteq X^{Env} \times 2^{Range(X)}$ where $(x, Range_{Env}(x)) \in L^{ag}$ for each $x \in X^{Env}$.*
• *Act^{Env} is $\bigcup_{ag \in Ag} Act^{ag} \cup$ Act–DY^{Intr}.*
• *ι^{Env}, P^{Env} and tr^{Env} are defined as on T-Env expanded over X^{Env}, L^{Env}, Act^{Env} and homomorphically instantiated.*

Iteration of instances and their systems.

Let $R, R', R'', \ldots$ be all protocol roles in the system. Assume $n_R, n_{R'}, n_{R''}, \ldots$ to be the number of instances of T-R, T-R', T-R'',..., respectively, present in the system. Consider m to be the total numbers of role-agents in the system. Then, the interactions over all agents $i \leqslant n_R$ of the (i, n_R, m)-concrete instances of T-R with a (n_R, m)-concrete environment and a (n_R, m)-concrete intruder, for each protocol-role T-R, define a concrete or product system of size $\overline{n} = (n_R, n_{R'}, n_{R''}, \ldots)$. This contains a concrete environment and a concrete intruder, which we denote $Env(\overline{n})$ and $Intr(\overline{n})$, respectively.

Atomic propositions for templates and agents.

Role and agent predicates. Let T-$R = (X, L \subseteq X \times A, \iota, Act, P, t)$ as above. Let AP_R be a set of atomic propositions over X and $\mathcal{V}_R : L \to \mathcal{P}(AP_R)$ be a valuation function of these atoms in T-R. Let ag be an instance of T-R. The above construction of ag over T-R induces a set AP_R^{ag} of atomic propositions over X^{ag}. Let $\mathcal{V}_R^{ag} : L^{ag} \to \mathcal{P}(AP_R^{ag})$ be a valuation function of these atoms in ag.

Let T-$Intr = (X', L' \subseteq X' \times A', \iota', Act', P', t')$ be as above. Let $AP_{T\text{–}Intr}$ be a set of atomic propositions over X'. From the definition of $Intr(n_R, m)$, we derive $AP^{Intr(n_R,m)}$ and $\mathcal{V}^{Intr(n_R,m)} : L^{Intr(n_R,m)} \to \mathcal{P}(AP^{Intr(n_R,m)})$. We do the same for $Env(n_R, m)$. On a concrete system, these are extended to $Env(\overline{n})$ and $Intr(\overline{n})$.

Let a product system be built as above and AP be the union of all its resulting atoms. Then, let $\mathcal{V}(\overline{n}) : \times_{1 \leqslant i \leqslant n_R} L^i \times \ldots \times L^{Intr(\overline{n})} \times L^{Env(\overline{n})} \to \mathcal{P}(AP)$ be defined as $\mathcal{V}(\overline{n})((l^1, \ldots, l^{Intr(\overline{n})}, l^{Env(\overline{n})})) = \{p \,|\, p \in V^a(l^a)\}$, for $a \in \{1, \ldots, Intr(\overline{n}), Env(\overline{n})\}$.

Concrete systems.

Definition 3.4 (Concrete security system). *Let R be a role, S be a PSS, and $\overline{n}$ be as above. A* concrete security system (CSS) of size $\overline{n}$ *for S is the tuple $S(\overline{n}) = (((ag_R^i)_{i \in \{1,\ldots,n_R\}})_R, Intr(\overline{n}), Env(\overline{n}), \mathcal{V})$.*

The tuple $S(\overline{n})$ given above is an interleaved interpreted system whose components are defined as in Sections 2 and 3.

The product of local objects (e.g., states, actions) in a CSS define global objects. The *unwound CSS* $\mathbb{S}(\overline{n})$ is the underlining Kripke structure obtained from a given CSS. The global actions and the global transition function induce paths over the global states of a CSS, as usually [16]. The indistinguishability relations over global states are: $\sim_{ag^i_R}$ for agents, $\sim_{Intr(\overline{n})}$ for the intruder, $\sim_{Env(\overline{n})}$ for the environment. These are defined on local equalities: $g \sim_j g'$ iff $g_j = g'_j$ where $j \in \{ag^i_R, Intr(\overline{n}), Env(\overline{n})\}$, and g_j is the local state of j in the global state g. This indistinguishability relation is correct cryptographically due to the fact that we model receiver-transparent protocols.

Specification language.

System properties are expressed in a variant of CTLK [16], stemming from the following BNF:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid EX\phi \mid E(\phi U \phi) \mid EG\phi \mid K_i\phi$$

where $i \in T\text{-}R, T\text{-}Intr$ and $p \in AP_R \cup AP_{Intr}$. Dual temporal modalities and the universal path quantifier A ("for all paths") are defined as usual [18]. These formulae are interpreted over any unwound interpreted system, like CSSs, as expected [16]; for formula satisfaction, we write $\mathbb{S}(\overline{n}) \models \phi$.

Since variables and atoms are indexed by roles, we now introduce a variant of *indexed CTLK*; we will use the *AG*-fragment of this.

Formula-schemata. We use abstract formula-schemata specified over a PSS. Applying instantiations to these, produces concrete formulae in the corresponding CSSs, irrespective of the numbers of concrete agents within. With our models, we only use formulae generated upon this *schema*:

$$[T\text{-}R : \vec{x}, \vec{u}, \ldots, T\text{-}Intr : \vec{w}]^+ AG(\phi(p(x,u), \ldots, p'(w), \ldots)),$$

where $\vec{x}$, $\vec{u}$, $\vec{w}$ are lists of variables, $x \in \vec{x}$, $u \in \vec{u}$, $w \in \vec{w}$, $p \in AP_R$, $p' \in AP_{T\text{-}Intr}$, ϕ is formed with the atoms p, p'. The values of the variables in $\vec{x}$ will iterate over the names of T-R instances, and elements of $\vec{u}$ and $\vec{w}$ will span values of variables inside atoms in AP_R and $AP_{T\text{-}Intr}$, respectively.

Concrete formulae. An *expansion of a schema inside a CSS* is the conjunction over all the 'names' a of agents iterated over with $\vec{x}$s, over all the values b inside their atoms iterated over with $\vec{u}$s, over all references to values c in atoms of $Intr$ iterated over with $\vec{w}$:

$$\bigwedge_{\vec{x}=\vec{a}, \vec{u}=\vec{b}, \ldots, \vec{w}=\vec{c}} AG(\phi(p^{ag}(a,b), \ldots, p^I(c), \ldots),$$

where $p^{ag} \in AP^{ag}_R$, $p^I \in AP^{Intr}$.

Indexed specifications. A formula following the above schema is called a $\vec{v}$*-indexed formula*, where $\vec{v}$ is called the *index* of the formula; it denotes all lists of variables bound to templates as per the schema. We use $\forall_{\vec{v}} AG\phi(\vec{v})$ to denote both an indexed schema in a PSS or its expansion in a CSS. By $\#(\vec{v})$ we refer to the vector $\#(\vec{v}) = (m_R, m_{R'}, \ldots)$ denoting the number m_R of variables in $\vec{v}$ bound to each template T-R; we assume w.l.o.g that $m_R \geqslant 1$ for each T-R.

Parameterised model checking and cut-offs of PSS.

Parameterised MC problem [21]. Given a PSS and an $\vec{v}$-indexed formula $\forall_{\vec{v}} AG\phi(\vec{v})$, the parameterised model checking problem (PMCP) concerns answering whether

$$\forall \overline{n} \geqslant \#(\vec{v}), \mathbb{S}(\overline{n}) \models \forall_{\vec{v}} AG\phi(\vec{v}),$$

The PMCP is undecidable [3] over general unbounded transition systems. For systems with communication patterns [21] similar to those of PSSs, the problem can be decided by finding a *cut-off*. This is the number of system components that suffices to be considered when evaluating a given specification.

Definition 3.5 (PSS Cutoff). *Let S be a PSS and $\vec{v}$ be an arbitrary index in an indexed formula. Let $\overline{c} = (c_R, c_{R'}, \ldots)$ be a vector of size equal to the number of protocol-roles R, R' in PSS, with $c_R, c'_R \geqslant 1$. The vector $\overline{c}$ is said to be a* PSS cutoff *if the following holds:*

$\mathbb{S}(\overline{c}) \models \forall_{\vec{v}} AG\phi(\vec{v})$ *iff* $\forall \overline{n} > \overline{c}.\, (\mathbb{S}(\overline{n}) \models \forall_{\vec{v}} AG\phi(\vec{v}))$,
where $\mathbb{S}(\overline{n})$ is S's unwound CSS of size $\overline{n} = (n_R, n_{R'}, \ldots)$.

To give a cut-off for PSS, we need the following notion. A *trivial assignment of variables for an index* $\vec{v}$ assigns different variables of the same sort (e.g., different nonces) to different values in their ranges. We can map these values to indices or natural numbers, since our ranges or our algebra support sets are infinitely countable; we use this in the next proofs. For $\forall_{\vec{v}} AG\phi(\vec{v})$ under a trivial assignment, we write $AG\phi(\vec{\epsilon})$.

Theorem 3.1 (Symmetry reduction). *Let S be a PSS and $\vec{v}$ be an arbitrary index in an indexed formula. For a $\vec{v}$-indexed formula $\forall_{\vec{v}} \phi(\vec{v})$, we have that*

$$\mathbb{S}(\overline{n}) \models \forall_{\vec{v}} \phi(\vec{v}) \text{ iff } \mathbb{S}(\overline{n}) \models AG\phi(\vec{\epsilon}),$$
where $\mathbb{S}(\overline{n})$ is S's unwound CSS of size $\overline{n} = (n_R, n_{R'}, \ldots)$

Proof. We adapt [15] for S, poising on receiver-transparency.

This lemma reduces the size of the formulae to check, making it sufficient to consider exactly one value from the domains of each of the variables. This is exploited next.

Finding cut-offs for PSSs.

Consider $\overline{c} = (c_R, c_{R'}, \ldots)$, where each c_R denotes as many agents as the number of variables admitted by T-R and appear in $\forall_{\vec{v}} AG\phi(\vec{v})$, plus 1 special agent; i.e., if $(m_R, m_{R'}, \ldots) = \#(\vec{v})$, then $c_R = m_R + 1$ for each protocol-role T-R. We show that $\overline{c}$ is a cutoff. To do this, we establish a relation between the CSS $\mathbb{S}(\overline{c})$ and an arbitrary CSS $\mathbb{S}(\overline{n})$ with $\overline{n} \geqslant \overline{c}$. More specifically, we define a function ζ_c that maps a path in $\mathbb{S}(\overline{c})$ to a path in $\mathbb{S}(\overline{n})$, and a function ζ_n that maps a path in $\mathbb{S}(\overline{n})$ to a path in $\mathbb{S}(\overline{c})$. Since the transition function in each system is triggered by one precise action at a time, any sequence $ga^1a^2\ldots$, where g is a concrete initial state and $a^1, a^2 \ldots$ are concrete actions, specifies a well-defined path. We begin with ζ_n which is defined as $\zeta_n(ga^1a^2\ldots) = g'a'^1a'^2\ldots$, where

- g' is any initial state in $\mathbb{S}(\overline{c})$ such that the intruder's *poss*-variables that refer to a concrete (special) agent $ag^{c_R}_R$, for any role R, are set to *true*;
- for each $i \geqslant 1$, if $a_i \in Act^{ag^x_R}$, where $x \geqslant c_R$, then a'_i is the null action;
- for each $i \geqslant 1$, if $a_i \in Act^{ag^x_R}$, where $x < c_R$, then a'_i is the action obtained from a_i by replacing each variable with index greater than c_R with c_R;
- for each $i \geqslant 1$, if a_i is an intruder's action, then a'_i is the action obtained from a_i by replacing each variable with index greater than c_R with c_R.

We now define ζ_c by $\zeta_c(g'a'^1a'^2\ldots) = ga^1a^2\ldots$, where

• g is an initial state in $\mathbb{S}(\overline{n})$ such that an intruder's *poss*-variable is set to *true* iff the variable is admitted by the intruder in $\mathbb{S}(\overline{c})$ and it is set to *true* in $\mathbb{S}(\overline{c})$;

• for each $i \geqslant 1$, $a^i = a'^1$.

On PSSs, we imposed that agents synchronise on actions. This is the core of the mappings above, which entail the following theorem, showing $\overline{c}$ to be our cutoff.

THEOREM 3.6. $\mathbb{S}(\overline{c}) \models \forall_{\vec{v}} AG\phi(\vec{v})$ *iff* $\mathbb{S}(\overline{n}) \models \forall_{\vec{v}} AG\phi(\vec{v})$ *for any* $\overline{n} \geqslant \overline{c}$.

PROOF. ($\Rightarrow$) Assume $\mathbb{S}(\overline{c}) \models \forall_{\vec{v}} AG\phi(\vec{v})$ and choose $\overline{n} \geqslant \overline{c}$. By Theorem 3.1, $\mathbb{S}(\overline{c}) \models AG\phi(\vec{\epsilon})$. Let π be an arbitrary path in $\mathbb{S}(\overline{c})$ originating from an initial state. We have that $(\mathbb{S}(\overline{c}), g) \models \phi(\vec{\epsilon})$ for every g in π. Consider $\zeta_c(\pi)$. By definition of ζ_c, $(\mathbb{S}(\overline{n}), g) \models \phi(\vec{\epsilon})$ for every g in $\zeta_c(\pi)$. Therefore, $\mathbb{S}(\overline{n}) \models AG\phi(\vec{\epsilon})$.

($\Leftarrow$) As above, but using ζ_n. □

The above provides a methodology to solve the PMCP for PSSs by checking every concrete systems up to the cutoff.

4. IMPLEMENTATION

We implemented the theoretical grounds of verifying parameterised security systems (PSS) in *MCMAS-S*, as an extension of the MCMAS-P model-checker [20]. We designed the input language for *MCMAS-S*, *SISPL*, tailored to our semantics.

A MAS-specification language for security protocols. Next, we use SISPL snippets. We start with an SISPL excerpt for the *A-role* in NSPK. Template for roles in SISPL include types of variables with denotation and range as per the security semantics:

```
Template Role_A
Protocol Vars: -- bound to the A-role
   ...
   n_a  :    {null,atom_a};
end Vars
Log Vars:  -- free in the A-role
   n_b  logs Role_B.n_b; ...
end Vars
Longterm Vars:  -- longterm data
   a    :    {null,alice}; ...
end Vars
```

Under `Log Vars`, *n_b* models the n_b free variable in NSPK's *A-role*. So, concrete values for *n_b* depend on the ranges defined in the *B*-role template (i.e., `n_b logs Role_B.n_b`). In a concrete system, with each new instance i of *Role_A*, a new value $atom\_a_i$ is added to the growing range of n_a, and each such instance will hold as many copies of n_b as there are instances of *Role_B*. To encode the operational semantics of actions, we designed several, overloaded shorthands to be used in a part called `Messaging Actions`:

```
...  Messaging Actions ...
  receive_byTransmit_msg2 composes (+n_a-,-n_b+,+a-);...
```

The keyword `composes` denotes that the concrete messages to be sent or received have ranges homomorphically defined via the ranges of the variables within. Then, the sign '+' in front of a variable denotes that the concrete agent has to have a non-null value for that variable in order for the action to be enabled or executable (see Section 2). A sign '-' in front of a variable stipulates that the action is executable irrespective of the value of this variable; due to `composes (...,-n_b...,)`, a *Role_A* agent in NSPK would receive the second message no matter the value of *n_b*. For this action, the '+' preceding *n_a* is used to implement matching-receive semantics: an instance of *Role_A* would not accept the second message in NSPK unless its inner value for n_a coincided with her a-priori value of *n_a*. Lastly, the sign '+' after a variable implements the setting nature of actions (see Section 2): `composes (...,n_b+...,)` denotes that the receipt of NSPK's second message by an instance of *Role_A* will assign a concrete value for *n_b* accordingly. The specification of protocol and evolution functions is minimalistic (as per Section 2) and uses PISPL [21] syntax, i.e., as in the MCMAS-P model-checker that we extended..

We now discuss the environment's SISPL specification.

```
Template Environment
Log Vars:
   a_log_protocol_msg1   logs Role_A.a;
   n_a_log_protocol_msg1 logs Role_A.n_a; ...
end Vars
Messaging Actions ...
-- A sends msg1, Env intercepts
sent_intercept_msg1   composes
   (-a_log_protocol_msg+,-n_a_log_protocol_msg1+,
    -b_log_protocol_msg1+);
-- Env's logs might be rewritten by Intr
tamper_log_msg1 composes
   (-a_log_protocol_msg1+,-n_a_log_protocol_msg1+,
    -b_log_protocol_msg1+); ...
```

The *Log Vars* part is inline with *T-Env* discussed in Section 2: in an instance-by-instance fashion, the environment stores the values for all the messages exchanged. In the `Messaging Actions` part, by the preceding '-' signs, we mean that the environment does not need any particular value to intercept messages; this will set all due values (i.e., the suffix + signs). The `tamper_log_msg1` lines show the *Env-Intr* synchronisation, where *Env*'s intercepted logs may be overwritten by Dolev-Yao compositions.

Now consider a snippet from the Intruder template:

```
... Messaging Actions ...
analz_msg_1_as_enc_pair composes
(-a_in_protocol_msg1+,-n_a_in_protocol_msg1+,
+b_in_protocol_msg1+);...
```

It is a part of the Dolev-Yao analysis on a NSPK message: if *Intr* knows the decrypting key (i.e., "`+b_in_protocol_msg1`"), then his variables will be set to the inner-values of this message (i.e., `n_a_in_protocol_msg1+`).

The specification of logical formulae is as in PISPL [21].

MCMAS-S: a prototype model checker for security protocols given as MAS specifications. The input to MCMAS-S is a PSS S and a set of logical formulae in indexed CTLK following our schemata, written as SISPL file. The model checker solves the PMCP for PSS. MCMAS-S is based on the same core parameterised model checking techniques as MCMAS-P [21]. In other words, it implements the techniques in Section 3, exploiting agent-environment synchronisation and symmetry reductions to find cut-offs as in Theorem 3.6. Once a cut-off $\overline{c}$ is calculated, the reachable state-space of $\mathbb{S}(\overline{c})$ is computed and encoded symbolically. Then, the formulae are checked on this and a result is presented.

MCMAS-S differs from MCMAS-P primarily in that agents can share variables and ranges. Like in MCMAS-P, the construction

of these ranges is done on-the-fly, as larger systems and formulae up to the cut-off $\bar{c}$ are being considered; however, the semantics in Section 3 implies that in MCMAS-S we expand these ranges differently with a security-driven fine-tuning in mind.

The C++ implementation of MCMAS-S is available at [1].

5. EXPERIMENTAL RESULTS

We evaluated our theoretical results and the implementation by verifying two authentication and key-establishment protocols: NSPK [27], presented in Section 2, and ARSPC [30]. The PSSs and the indexed CTLK formulas to express their respective authentication and secrecy goals were coded in a SISPL file.

Following Section 2, NSPK's authentication goal is

$$AG(end_A(\vec{u},\vec{v}) \wedge end_B(\vec{u'},\vec{v'}) \rightarrow agree\_naAB(\vec{u},\vec{u'},\vec{v},\vec{v'},\vec{w},\vec{w'})),$$

where $end_A \in AP_A$ denotes that template T-A of A-role is at the end-step of the protocol, $\vec{u}$ and $\vec{v}$ range over instances of T-A, $agree\_naAB$ is a conjunction of atoms in AP^A and AP^B expressing that the values of A, n_A and B are the same in (instances of) T-A and T-B, and $\vec{w}$ iterates over the values of A, n_A and B inside T-A. Similarly, $\vec{u'}$, $\vec{v'}$ span instances of T-B and $\vec{w'}$ ranges over the values of the cited variables inside T-B.

Following Section 2, NSPK's secrecy goal is

$$AG(end_A(\vec{u},\vec{v}) \wedge end_B(\vec{u'},\vec{v'}) \rightarrow \neg K_{Intr}(holds\_na(\vec{u},\vec{u'},\vec{v},\vec{v'}\vec{w},\vec{w'}))),$$

where $holds\_na$ is a conjunction of atoms in AP_A and AP_B expressing that the value of n_A is the same in (instances of) T-A and T-B, and the rest is as previously explained.

By using MCMAS-S, we obtain a cut-off of $(2, 2)$ underpinning an attack which refuted the first formula above. This attack consists of two A-parties and two B-parties, with the intruder tampering with the sessions of these parties.

Indeed, for the first formula, we also find a counterexample emulating the famous Lowe-attack [24] on NSPK. For the second formula, MCMAS-S offers us a longer counterexample but analogous to that of Lowe's attack, showing that the intruder learns the value of n_a in an instance of T-A and replays it to an instance of T-B, with $\neg K_{Intr}(holds\_na(\ldots))$ failing therein.

Many authentication and key-establishment protocols have inter-session attacks similar to Lowe's attack in NSPK. Indeed, ARSPC (Andrew Secure RPC) [30] is a key-establishment protocol where two concurrent sessions can be exploited in this way [25]. To avoid this, Lowe proposed [25] a strengthening [22] of ARSPC.

We used MCMAS-S to check this version of ARSPC against a CTLK formula similar to NSPK's authentication-requirement but expressed for the key established in ARSPC. We found a cut-off of $(2, 2)$ as above; by checking the concrete models, we were able to establish that Lowe's version of ARSPC is correct in the unbounded setting.

On a PC with an Intel(R) Core(TM)2 Quad CPU Q8200 2.33GHz, with 4GB of total memory, running Ubuntu 3.13.0-55-generic, it takes approximately 5 seconds to find the cut-offs above and to check the corresponding systems. The SISPL files for these tests are available together with MCMAS-S at [1].

6. CONCLUSIONS

Forthcoming pervasive applications in the IOTcall for the deployment of MAS to be secure. To this end, this paper puts forward a technique to verify whether the security protocols underlying these applications are correct in a provable way. Concretely, we introduced a semantics that accounts for an unbounded number of protocol sessions and unbounded size data-exchanges under a Dolev-Yao threat and we defined the parameterised model checking problem for these. Though the problem is generally undecidable, in our case we solved it by analysing a finite number of bounded systems. This was possible due to our models enjoying agent-environment synchronisations and a certain symmetry, which is in part down to he class of protocols considered. We mechanised these in MCMAS-S: a model checker which verifies parameterised MAS modelling security protocols against requirements given a fragment of CTLK. We verified versions of the NSPK and ARSPC authentication and key-establishment protocols [31], and we proved their (in)security in an unbounded-session setting.

Related Work. We built upon [20, 21], where the parameterised model checking problem for MAS and the model checker MCMAS-P were introduced. However, neither the semantics nor the tool in [20] can account for security models. For instance, in [20] there is no support for multiple templates, partitioned variables in templates, or for unbounded variables and ranges inside the agents; these are needed to express protocol-roles and the unbounded ranges of, e.g., long-term keys.

The Dolev-Yao-based [14] security semantics is widely adopted in security-verification [5, 29, 12]. But few of these [7] support agent-driven models and temporal-epistemic specifications. Unlike [7], we consider an unbounded number of protocol sessions and agents therein, and so we check a protocol in general terms rather than just draw conclusions for a specific number of principals and sessions of its exchanges.

ProVerif and Tamarin are tools for security-verification in unbounded settings [26, 6], based on methods other than model-checking. They are semidecidable, halting only when an attack is found. This is not the case of MCMAS-S, which ascertains both correctness and incorrectness of unbounded systems. Moreover, those tools generally cannot handle state-based properties in unbounded-size models. And even when they can do so in the bounded setting [4], their requirements cannot pertain to agency-based logics, so they cannot reason on the epistemic states of the principals explicitly, as we do here. As such, it has often been argued that using primitives on agents' knowledge can simplify and clarify specifications of security protocols [2, 17, 11]. Indeed, note that the epistemic formulae we verify, $\neg K_{Intr}(\ldots)$), can be used à la [8] to express privacy requirements via the attacker's ignorance w.r.t. to facts spanning the whole state-space of the systems.

Future Work. We will extend the approach to other classes of protocols and exhibit the analysis of privacy requirements, given their intrinsic epistemic nature.

7. ACKNOWLEDGEMENTS

This research was funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 661362 and by the EPSRC under grant EP/I00520X and a Doctoral Prize Scholarship.

REFERENCES

[1] MCMAS-S. `http://vas.doc.ic.ac.uk/software/extensions/`, 2016.

[2] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proc. of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91)*, pages 201–216, 1991. ACM Press.

[3] K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.

[4] D. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155, 2015. ACM.

[5] D. Basin, S. Modersheim, and L. Vigano. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[6] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc.of the 14th IEEE Workshop on Computer Security Foundations (CSFW'01)*, pages 82, 2001. IEEE Computer Society.

[7] I. Boureanu, M. Cohen, and A. Lomuscio. A compilation method for the verification of temporal-epistemic properties of cryptographic protocols. *Journal of Applied Non-Classical Logics*, 19(4):463–487, 2009.

[8] I. Boureanu, A. V. Jones, and A. Lomuscio. Automatic verification of epistemic specifications under convergent equational theories. In *Proc. of the 11th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'12)*, pages 1141–1148. IFAAMAS Press, 2012.

[9] M. Brown, B. An, C. Kiekintveld, F. Ordóñez, and M. Tambe. An extended study on multi-objective security games. *Autonomous Agents and Multi-Agent Systems*, 28(1):31–71, 2014.

[10] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[11] R. Chadha, S. Delaune, and S. Kremer. Epistemic logic for the applied pi calculus. In *Formal Techniques for Distributed Systems*, volume 5522 of *LNCS*, pages 182–197. Springer Berlin Heidelberg, 2009.

[12] B. Conchinha, D. Basin, and C. Caleiro. FAST: an efficient decision procedure for deduction and static equivalence. In *Proc. of the 22nd International Conference on Rewriting Techniques and Applications (RTA'11)*, pages 11–20, 2011.

[13] J. Dickerson, J. Goldman, J. Karp, A. Procaccia, and T. Sandholm. The computational rise and fall of fairness. In *Proc. of the 28th AAAI Conference on Artificial Intelligence*, pages 1405–1411, 2014.

[14] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactionson Information Theory 29*, 29(2):198–208, 1983.

[15] E. Emerson and A. Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1):105–131, 1996.

[16] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[17] J. Y. Halpern and R. Pucella. Modeling Adversaries in a Logic for Security Protocol Analysis. In *Proc. of the Workshop on Formal Aspects of Security (FASec02)*, volume 2629 of *LNCS*, pages 115–132. Springer-Verlag, 2002.

[18] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems (2nd edition)*. Cambridge University Press, Cambridge, England, 2004.

[19] J. Jennings, G. Whelan, and W. Evans. Cooperative search and rescue with a team of mobile robots. In *Proc. of the 8th International Conference on Advanced Robotics (ICAR'97)*, pages 193–200, 1997.

[20] P. Kouvaros and A. Lomuscio. A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 2013–2019. AAAI Press, 2013.

[21] P. Kouvaros and A. Lomuscio. Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189, 2016.

[22] Laboratoire Spécification et Vérification (LSV), École Normale Supérieure Cachan. Lowe's version of the ARSPC protocol. A Library of Cryptographic Protocols Descriptions.

[23] A. Lomuscio, W. Penczek, and H. Qu. Partial order reductions for model checking temporal-epistemic logics over interleaved multi-agent systems. *Fundamenta Informaticae*, 101(1):71–90, 2010.

[24] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, Nov. 1995.

[25] G. Lowe. Some new attacks upon security protocols. In *Proc. of the 9th IEEE Workshop on Computer Security Foundations (CSFW'96)*, pages 162, 1996. IEEE Computer Society.

[26] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proc. of the 25th International Conference on Computer Aided Verification (CAV'13)*, pages 696–701, Berlin, Heidelberg, 2013. Springer-Verlag.

[27] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *ACM Communications*, 21(12):993–999, Dec. 1978.

[28] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1-2):85–128, Jan. 1998.

[29] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. *Proc. of the 21st IEEE Symposium on Security and Privacy (S&P'01)*, 2001.

[30] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Trans. Comput. Syst.*, 7(3):247–280, Aug. 1989.

[31] SPORE. Security protocols open repository. http://www.lsv.ens-cachan.fr/spore.