# Partial Order Reduction for Branching Security Protocols

Wan Fokkink
*VU Amsterdam*

Mohammad Torabi Dashti
*ETH Zürich*

Anton Wijs
*TU Eindhoven*

***Abstract*—Two extensions of the partial order reduction algorithm of Clarke, Jha and Marrero are presented. The proposed algorithms are suitable for branching security protocols, e.g. optimistic fair contract signing schemes. The first extension is proved to generate a reduced state space which is branching bisimilar to the full state space, while the second extension generates a state space that is trace equivalent to the full state space. Experimental results using an implementation of the algorithms in the toolset of the $\mu$CRL process algebra are reported.**

## I. Introduction

Two major approaches to automatic verification of security protocols are model checking and constraint solving. Both techniques in principle need to enumerate all possible interleavings of actions performed by protocol participants. Partial order reduction (POR) techniques identify and avoid generating identical interleavings, modulo the properties that are to be verified, to reduce the time and memory used in verification.

Standard POR algorithms for communication protocols, e.g. as implemented in SPIN, are in general not effective for security protocols (see section VIII for details). Clarke, Jha and Marrero (CJM) [10], [12] were the first to formally present a POR algorithm for security protocols and determine the class of modal properties that are preserved by it. They observe that the knowledge of the Dolev-Yao attacker model in the course of each protocol run is non-decreasing, and, intuitively, with more knowledge the attacker can do more (harm). Therefore, when verifying security protocols which yield finite-depth executions, in the presence of the Dolev-Yao attacker, it is safe to prioritize actions that increase the attacker's knowledge over other actions. This is the heart of the POR algorithm of [10], which is used in BRUTUS [11], a model checker tailored for security protocols. This algorithm has also been considered by Millen and Shmatikov [23] as a means to reduce the number of constraint sets that have to be solved to verify a security protocol.

The POR algorithm of CJM assumes that the participants of security protocols are *non-branching*, meaning that each participant at each state of the protocol has at most one single action to perform. This assumption has been widely used in the literature for modelling various authentication and key distribution schemes since the early years of security protocol analysis, e.g. see the "ping-pong" protocols of [15]. In practice, however, participants of e.g. authentication protocols may have *choice points*. They can for instance take alternative actions when a received message does not match a certain pattern, or when a timeout occurs. Therefore, any faithful model of these protocols must allow such choice points in the specification as well. More importantly, a large number of security protocols explicitly prescribe more than one possible behavior for the participants. Prominent examples are optimistic fair exchange protocols, including fair non-repudiation, fair payment, certified email, and electronic contract signing protocols, e.g. see [4]. Participants of these protocols can choose between continuing the normal flow of the protocol and resorting to a trusted entity, in case of long delays of the opponent or communication failures. These protocols can be properly modelled only if choice points are allowed in the specification of their participants.

Concerning properties, the POR algorithm of Clarke et al. preserves a variant of past-time linear-time temporal logic (LTL), augmented with knowledge operators. LTL is in general insufficient for branching security protocols: A natural feature of branching protocols is that they aim at branching-time security properties. Prominent examples of branching-time security requirements are various properties of fair exchange protocols which can be expressed in alternating-time temporal logic [22], security requirements of Web services that are naturally stated in logics based on computation-tree temporal logic (CTL) [28], [29], anonymity [8], and information flow control properties [13].

In this paper, we report on extensions of the POR algorithm of CJM to handle security protocols in which participants may have choice points. Two POR algorithms are presented for this purpose. The first one, which is called *reducing* algorithm, is proved to generate a reduced state space that is branching bisimilar [18] to the full state space. This reduction algorithm is thus suitable for checking branching-time security properties [24]. Our proposed bisimulation relation can be of independent interest, e.g., for establishing relations among protocol models which inherently allow branching or admit partial order interpretations, such as the applied $\pi$-calculus [1] and the strand space model [31].

The second algorithm, which is called *pruning* algorithm, is proved to generate a state space that is trace equivalent to the full state space. This algorithm is thus suitable for checking linear-time security properties.

The proposed POR algorithms have been implemented in

the $\mu$CRL toolset [7], which is based on process algebra and abstract data types [19]. We report on some experimental results for an optimistic fair exchange protocol using the $\mu$CRL toolset.

*Structure of the paper:* Section II introduces the preliminaries. Security protocols and their execution semantics are described in section III. There we introduce the "full" execution model along with our proposed "reduced" and "pruned" execution semantics. (The naming convention *reduced* versus *pruned* is merely a means for distinguishing the two execution models.) In section IV it is proved that for security protocols the reduced state space is branching bisimilar to the full one. Section V shows that for security protocols the pruned state space is a so-called *ample* substate space of the full state space, which implies that they are trace equivalent. The proposed execution semantics are implemented as POR algorithms using the notion of *ample sets*, in section VI. Section VII evaluates the effectiveness of the proposed POR algorithms in the context of a case study. Related work is discussed in section VIII, while section IX concludes the paper.

## II. Preliminaries

A *directed acyclic graph* (dag) is a tuple $T = (S, r, E)$, where $S$ is a set of *states*, $r \in S$ is the *root*, and $E$ is a set of *events* (or *actions*). To each event $e \in E$, we associate a subset of $S \times S$. We use $e$ and the subset of $S \times S$ associated to $e$ interchangeably when confusion is unlikely, and for all $e \in E$ assume that (1) $\forall s \in S.\ (s, s) \notin E^+$, and (2) $\forall s \in S.\ (r, s) \in E^*$. Here $E^+$ and $E^*$ are the transitive and the transitive reflexive closures of the relations in $E$, respectively. As a convention, we write $s \xrightarrow{e} s'$ if $(s, s') \in e$, and define the set of events *enabled* at state $s$ in dag $T$ as $en^T(s) = \{e \in E \mid \exists s' \in S.\ s \xrightarrow{e} s'\}$. We may write $en(s)$ for $en^T(s)$ when $T$ is clear from the context. Dag $T$ is *single-image* if each event in $E$ is a partial function. For a single-image dag we write $e(s) = s'$ iff $(s, s') \in e$. Dag $T' = (S', r', E')$ is a *subdag* of $T = (S, r, E)$ iff $S' \subseteq S$, $r' = r$ and $E' \subseteq E$.

A finite *path*, or *trace*, in $T$ is a finite sequence $\sigma = s_0, s_1, \cdots, s_k$, such that there exist events $e_0, \cdots, e_{k-1}$ with $\forall 0 \leq i < k.\ s_i \xrightarrow{e_i} s_{i+1}$. We say $\sigma$ is *rooted* at $s_0$ and may write $s_0 \xrightarrow{e_0 \cdots e_{k-1}} s_k$. Note that if $T$ is single-image, then the sequence of events $e_0, e_2, \cdots, e_{k-1}$ uniquely determines trace $\sigma$, if the root of the trace is given. The *depth* of a state $s \in S$ in $T$ is the length of a shortest trace $\sigma = s_0, \cdots, s$ that is rooted at $r$, i.e. $s_0 = r$. A dag is of *finite depth* if there is a finite upper bound on the depths of the states in the dag.

Let $U$ be a set of symbols, and $W \subseteq U$. For $u \in U^*$, the sequence that is obtained by removing all elements of $W$ from $u$ is denoted $\mathsf{s}_W(u)$. For two sequences $u, v \in U^*$ we write $u \overset{W}{\equiv} v$ iff $\mathsf{s}_W(u) = \mathsf{s}_W(v)$. Clearly $\overset{W}{\equiv}$ is an equivalence relation on $U^*$.

**Definition 1** (Ample subdags)**.** *Let $W \subseteq E$ be a set of events in $T = (S, r, E)$. A subdag $T' = (S', r', E')$ of $T$ is an* ample *subdag modulo $W$ if for each trace $\sigma$ rooted at $r$ in $T$ there exists at least one trace $\sigma'$ rooted at $r'$ in $T'$ such that $\sigma \overset{W}{\equiv} \sigma'$.*

Under suitable transformations on Kripke structures or labeled transition systems (which are generalizations of our dags), the linear-time temporal logic $\mathrm{LTL}_{-\mathrm{X}}$ is preserved under the ample subdag relation [26].

**Definition 2** (Branching bisimulation [18])**.** *Let $W \subseteq E$ be a set of events in $T = (S, r, E)$, called* silent *events. A binary relation $R$ on the set of states $S$ is a* branching bisimulation *relation modulo $W$ iff*

- *$R$ is symmetric, and*
- *$(p, q) \in R$ and $p \xrightarrow{e} p'$ implies that either $e \in W$ and $(p', q) \in R$, or there exist $\hat{q}$ and $q'$ such that $q \xrightarrow{e_1 \cdots e_m} \hat{q}$ and $\hat{q} \xrightarrow{e} q'$, for some $m \geq 0$ and $e_1, \cdots, e_m \in W$, while $(p, \hat{q}) \in R$ and $(p', q') \in R$.*

*Two states are branching bisimilar if there exists a branching bisimulation that relates them.*

Branching bisimilarity is an observational equivalence which ignores silent events while preserving the branching structure of systems. Under suitable transformations on Kripke structures or labeled transition systems, the branching-time temporal logic $\mathrm{CTL}^*_{-\mathrm{X}}$ is preserved under branching bisimilarity [24].

## III. Security protocols

In this section we give an abstract model for security protocols.

A security protocol consists of a finite set of participants $\Pi = \{1, \cdots, n\}$, with $n \geq 1$. Each participant $i \in \Pi$ is modelled as a dag $T_i = (S_i, r_i, E_i)$ with the following properties:

- $T_i$ is single-image.
- $T_i$ is of finite depth.
- The events in $E_i$ are *communication* actions, i.e. each $e \in E$ is of the form $snd_i(m)$ or $rcv_i(m)$, where *message* $m$ belongs to $Msg$, as defined below.

The set of messages $Msg$ is the set of ground terms induced by a free term algebra $\Sigma$; that is $M = \mathcal{T}_\Sigma$. We write $\mathsf{Snd} = \{snd_i(m) \mid i \in \Pi, m \in Msg\}$ and $\mathsf{Rcv} = \{rcv_i(m) \mid i \in \Pi, m \in Msg\}$, when $\Pi$ is clear from the context. Note that

$$\forall i, j \in \Pi.\ i \neq j \implies E_i \cap E_j = \emptyset \tag{1}$$

The function $[\cdot] : (\cup_{j \in \Pi} E_j) \to \Pi$ is defined as $[snd_i(m)] = i$ and $[rcv_i(m)] = i$.

We remark that participants of security protocols are typically specified as finitely-branching dags of finite depth in which messages appearing in the events contain variables; that is $M = \mathcal{T}_{\Sigma(\mathcal{V})}$, with $\mathcal{V}$ being a countable set of

variables. Such "symbolic" representations yield a single-image infinitely-branching dag of finite depth, by simply substituting variables with all possible message contents. The correspondence between our "explicit" model versus "symbolic" models is thus immediate.

Internal actions, i.e. non-communication actions, are excluded from our model, in order to avoid unnecessary clutter in the proofs. The formalization can be extended to accommodate internal actions in a straightforward manner; see section VI.

Security protocols are executed in the presence of an *attacker*. We do not model the attacker explicitly as a dag; instead the attacker's abilities are reflected in our protocol execution models. The attacker has a set of inference rules which enable him to compose new messages from the messages he knows. We write $\mathfrak{C}(M)$ for the set of messages that the attacker can derive from $M \subseteq Msg$. The only property of function $\mathfrak{C}$ that is relevant for our results is *monotonicity*:

**Monotonicity**: $M \subseteq M' \implies \mathfrak{C}(M) \subseteq \mathfrak{C}(M')$

### A. Protocol executions

A security protocol can be *executed* in the presence of the attacker with initial knowledge $K_0$, creating an *execution dag* for the protocol. We attribute an asynchronous message passing semantics to security protocols in which messages are passed through the attacker; the attacker can in particular destroy the transmitted messages. The execution dag of a protocol consisting of participants $\Pi = \{1, \cdots, n\}$ is $T_\Pi = (S, r, E)$ where $S \subseteq S_1 \times \cdots \times S_n \times 2^{Msg}$ and $E$ are the smallest sets satisfying:

- $r = (r_1, \cdots, r_n, K_0),\ r \in S$.
- Let $z_1 = (s_1, \cdots, s_i, \cdots, s_n, K) \in S$.
  - If $s_i \overset{snd_i(m)}{\rightarrow} s_i'$ for some $i \in \Pi$, then $z_2 = (s_1, \cdots, s_i', \cdots, s_n, K \cup \{m\}) \in S$. In this case $snd_i(m) \in E$ and $(z_1, z_2) \in snd_i(m)$.
  - If $s_i \overset{rcv_i(m)}{\rightarrow} s_i'$, for some $i \in \Pi$, and $m \in \mathfrak{C}(K)$, then $z_2 = (s_1, \cdots, s_i', \cdots, s_n, K) \in S$. In this case $rcv_i(m) \in E$ and $(z_1, z_2) \in rcv_i(m)$.

The following lemma is immediate by the definition of $T_\Pi$ and condition 1.

**Lemma 1.** *If $T_i$ is single-image for all $i \in \Pi$, then $T_\Pi$ is single image. If $T_i$ is of finite depth for all $i \in \Pi$, then $T_\Pi$ is of finite depth.*

Given an execution dag $T_\Pi = (S, r, E)$, for $z = (s_1, \cdots, s_n, K) \in S$ we define $en_i(z) = en^{T_i}(s_i)$ for $i \in \Pi$. Note that in general $en_i(z) \neq en(z) \cap E_i$. By the definition of $T_\Pi$ it is immediate that for all $z \in S$ and $i \in \Pi$, $en_i(z) \cap \mathsf{Snd} \subseteq en(z) \cap \mathsf{Snd}$. However, if $en_i(z)$ contains $rcv_i(m)$ events, with $m \notin \mathfrak{C}(K)$ and $K$ being the knowledge of the attacker at $z$, then $en_i(z) \cap \mathsf{Rcv} \not\subseteq en(z) \cap \mathsf{Rcv}$. Therefore, in general $\cup_{i \in \Pi} E_i \cap \mathsf{Snd} \not\subseteq E \cap \mathsf{Snd}$.

The following lemma intuitively states that events of different participants are "independent". This is because the communication events occur only between a participant and the attacker.

**Lemma 2** (Independence property)**.** *Let $T_\Pi = (S, r, E)$ be the execution dag of a security protocol for the set of participants $\Pi = \{1, \cdots, n\}$. For any $z_1, z_2 \in S$, if $(z_1, z_2) \in e$, then $en_j(z_1) = en_j(z_2)$ for all $j \neq [e]$.*

*Proof:* Immediate by the definition of $T_\Pi$. ∎

We remark that generally $(z_1, z_2) \in e$ does *not* imply $en(z_1) \cap E_j = en(z_2) \cap E_j$ for all $j \neq [e]$. Namely, in $z_2$ the knowledge of the attacker may have increased (compared to $z_1$), hence enabling more receive events for the participants.

The following lemma informally establishes the "diamond" property in execution dags, i.e. if $e_a(s) = s_a$ and $e_b(s) = s_b$, where $e_a$ and $e_b$ are events of different participants and $s$ is an arbitrary state, then $e_b(s_a) = e_a(s_b)$.

**Lemma 3** (Confluence property)**.** *Let $T_\Pi = (S, r, E)$ be the execution dag of a security protocol for the set of participants $\Pi = \{1, \cdots, n\}$. For any $z_1, z_2, z_3 \in S$, if $(z_1, z_2) \in e$, with $e \in E_i$ for some $i \in \Pi$, and $z_1 \overset{\mathfrak{e}_1 \cdots \mathfrak{e}_\ell}{\rightarrow} z_3$ with $\mathfrak{e}_j \notin E_i$ for all $j = 1, \cdots, \ell$ (where $\ell \geq 0$), then $\exists z_4 \in S$ such that $z_2 \overset{\mathfrak{e}_1 \cdots \mathfrak{e}_\ell}{\rightarrow} z_4$ and $(z_3, z_4) \in e$.*

*Proof:* The proof is by induction on $\ell$. If $\ell = 0$, then $z_1 = z_3$ and $z_2 = z_4$. Clearly $(z_3, z_4) \in e$. Now assume the claim holds for $\ell = m$; below, we prove it for $\ell = m + 1$.

Suppose $(z_1, z_2) \in e$, $e \in E_i$, $z_1 \overset{\mathfrak{e}_1 \cdots \mathfrak{e}_m}{\rightarrow} z_3'$, $z_3' \overset{\mathfrak{e}_{m+1}}{\rightarrow} z_3$, and $\mathfrak{e}_1 \cdots \mathfrak{e}_{m+1} \notin E_i$. Due to the induction hypothesis, there exists a $z_4'$ such that $(z_3', z_4') \in e$, and $z_2 \overset{\mathfrak{e}_1 \cdots \mathfrak{e}_m}{\rightarrow} z_4'$. Since $\mathfrak{e}_{m+1} \notin E_i$, the independence property of $T_\Pi$ and monotonicity of $\mathfrak{C}$ imply that there exists a $z_4 \in S$ such that $z_3 \overset{e}{\rightarrow} z_4$ and $z_4' \overset{\mathfrak{e}_{m+1}}{\rightarrow} z_4$. Now, clearly we have $z_2 \overset{\mathfrak{e}_1 \cdots \mathfrak{e}_{m+1}}{\rightarrow} z_4$. This completes the proof. ∎

### B. Reduced protocol executions

In this section, we construct a *reduced* execution dag for security protocols which is a subdag of $T_\Pi$. The idea of the construction is that at each state $z = (s_1, \cdots, s_n, K)$ we check if any of the processes of $\Pi$ only performs a single $\mathsf{Snd}$ event. If so, only the single event of that process is further *explored*. The key observation here is that the knowledge set of the attacker is increased by $\mathsf{Snd}$ events, and this can only enable more $\mathsf{Rcv}$ events. The independence and confluence properties of the execution dags allow us to only explore such "knowledge increasing" events. The condition on the process being able to perform only a single $\mathsf{Snd}$ event pertains to preserving branching-time properties in POR algorithms (cf. [17]). The formal definition is given below.

Consider a protocol consisting of participants $\Pi = \{1, \cdots, n\}$, executed in the presence of the attacker with initial knowledge $K_0$. The *reduced* execution dag for the protocol is $\mathfrak{T}_\Pi = (S, r, E)$ where $S \subseteq S_1 \times \cdots \times S_n \times 2^{Msg}$ and $E$ are the smallest sets satisfying:

- $r = (r_1, \cdots, r_n, K_0),\ r \in S$.
- Let $z_1 = (s_1, \cdots, s_i, \cdots, s_n, K) \in S$ and define $J = \{j \in \Pi \mid |en_j(z_1)| = 1 \wedge en_j(z_1) \subseteq \mathsf{Snd}\}$. If $J \neq \emptyset$, then let $I = \{i \mid i = \min J\}$, else let $I = \Pi$. That is, either $I$ is a singleton, or $I = \Pi$.
  - If $s_i \overset{snd_i(m)}{\rightarrow} s_i'$, for some $i \in I$, then $z_2 = (s_1, \cdots, s_i', \cdots, s_n, K \cup \{m\}) \in S$. In this case $snd_i(m) \in E$ and $(z_1, z_2) \in snd_i(m)$.
  - If $s_i \overset{rcv_i(m)}{\rightarrow} s_i'$, for some $i \in I$, and $m \in \mathfrak{C}(K)$, then $z_2 = (s_1, \cdots, s_i', \cdots, s_n, K) \in S$. In this case $rcv_i(m) \in E$ and $(z_1, z_2) \in rcv_i(m)$.

The following lemma is immediate by the definitions of $T_\Pi$ and $\mathfrak{T}_\Pi$.

**Lemma 4.** *Let $\Pi$ be the set of participants of a security protocol, and $T_\Pi = (S, r, E)$ and $\mathfrak{T}_\Pi = (S', r', E')$ be the execution dag and the reduced execution dag of the protocol, respectively.*

1) *$\mathfrak{T}_\Pi$ is a subdag of $T_\Pi$.*
2) *For all $z \in S'$, $en^{\mathfrak{T}_\Pi}(z) = \emptyset$ only if $en^{T_\Pi}(z) = \emptyset$.*

The following proposition shows that, intuitively, events enabled in $T_\Pi$ may be delayed in $\mathfrak{T}_\Pi$, but they are not indefinitely ignored. Below, we let $T_\Pi = (S, r, E)$ and $\mathfrak{T}_\Pi = (S', r', E')$; moreover $en^T(z)$ refers to $en^{T_\Pi}(z)$, and $en^{\mathfrak{T}}(z')$ refers to $en^{\mathfrak{T}_\Pi}(z')$.

**Proposition 1.** *Let $z \in S'$ and $e \in en^T(z)$. Then there exist a $z' \in S'$ and $\ell \geq 0$ such that $z \overset{\mathfrak{e}_1 \cdots \mathfrak{e}_\ell e}{\rightarrow} z'$ in $\mathfrak{T}_\Pi$, where $\forall 1 \leq j \leq \ell.\ \mathfrak{e}_j \in \mathsf{Snd} \wedge [\mathfrak{e}_j] \neq [e]$.*

*Proof:* We observe that $\mathfrak{T}_\Pi$ is of finite depth, due to lemmas 1 and 4. Therefore all paths in $\mathfrak{T}_\Pi$ rooted at $z$ are either of the form $z_0, \cdots, z_\ell$, such that $z_0 = z$ and $en^{\mathfrak{T}}(z_\ell) = \emptyset$, or can be extended to such a path. Take any such extended path. We claim that for all $0 \leq i < \ell$:

$$(\exists 0 \leq j \leq i.\ e \in en^{\mathfrak{T}}(z_j)) \vee e \in en^T(z_{i+1}) \qquad (2)$$

Assume $e \notin en^{\mathfrak{T}}(z_j)$ for all $0 \leq j \leq i$. Then, due to the definition of $\mathfrak{T}_\Pi$, $z_0 \overset{\mathfrak{e}_0 \cdots \mathfrak{e}_i}{\rightarrow} z_{i+1}$, and for all $0 \leq j \leq i$ we have $[\mathfrak{e}_j] \neq [e]$ and $\mathfrak{e}_j \in \mathsf{Snd}$. Now, by the confluence property we have $e \in en^T(z_{i+1})$.

Due to lemma 4, $en^{\mathfrak{T}}(z_\ell) = \emptyset \implies en^T(z_\ell) = \emptyset$. This, along with relation (2), implies that there exists a $0 \leq j < \ell$ such that $e \in en^{\mathfrak{T}}(z_j)$. ∎

We remark that lemma 1 would fail if participants of security protocols were defined as dags of infinite depth.

### C. Pruned protocol executions

In this section, we construct a *pruned* execution dag for security protocols, which is a subdag of $T_\Pi$. The idea of the construction is that at each state $z = (s_1, \cdots, s_n, K)$ we check if any of the processes of $\Pi$ only performs $\mathsf{Snd}$ events; this need not be a singleton, in contrast to the reduced dags of section III-B. If so, only the events of that process are further explored. The formal definition is given below.

Consider a protocol consisting of participants $\Pi = \{1, \cdots, n\}$ executed in the presence of the attacker with initial knowledge $K_0$. The *pruned* execution dag for the protocol is $\mathfrak{P}_\Pi = (S, r, E)$ where $S \subseteq S_1 \times \cdots \times S_n \times 2^{Msg}$ and $E$ are the smallest sets satisfying:

- $r = (r_1, \cdots, r_n, K_0),\ r \in S$.
- Let $z_1 = (s_1, \cdots, s_i, \cdots, s_n, K) \in S$ and define $J = \{j \in \Pi \mid |en_j(z_1)| \geq 1 \wedge en_j(z_1) \subseteq \mathsf{Snd}\}$. If $J \neq \emptyset$, then let $I = \{i \mid i = \min J\}$, else let $I = \Pi$. That is, either $I$ is a singleton, or $I = \Pi$.
  - If $s_i \overset{snd_i(m)}{\rightarrow} s_i'$, for some $i \in I$, then $z_2 = (s_1, \cdots, s_i', \cdots, s_n, K \cup \{m\}) \in S$. In this case $snd_i(m) \in E$ and $(z_1, z_2) \in snd_i(m)$.
  - If $s_i \overset{rcv_i(m)}{\rightarrow} s_i'$, for some $i \in I$, and $m \in \mathfrak{C}(K)$, then $z_2 = (s_1, \cdots, s_i', \cdots, s_n, K) \in S$. In this case $rcv_i(m) \in E$ and $(z_1, z_2) \in rcv_i(m)$.

Note that the only difference between $\mathfrak{P}_\Pi$ and $\mathfrak{T}_\Pi$ lies in the definition of the set $J$. Namely, $J = \{j \in \Pi \mid |en_j(z_1)| \geq 1 \wedge en_j(z_1) \subseteq \mathsf{Snd}\}$ in $\mathfrak{P}_\Pi$, while $J = \{j \in \Pi \mid |en_j(z_1)| = 1 \wedge en_j(z_1) \subseteq \mathsf{Snd}\}$ in $\mathfrak{T}_\Pi$.

The following lemma is immediate by the definitions of $T_\Pi$ and $\mathfrak{P}_\Pi$.

**Lemma 5.** *Let $\Pi$ be the set of participants of a security protocol, and $T_\Pi = (S, r, E)$ and $\mathfrak{P}_\Pi = (S', r', E')$ the execution dag and the pruned execution dag of the protocol, respectively.*

1) *$\mathfrak{P}_\Pi$ is a subdag of $T_\Pi$.*
2) *For all $z \in S'$, $en^{\mathfrak{P}_\Pi}(z) = \emptyset$ only if $en^{T_\Pi}(z) = \emptyset$.*

## IV. A BRANCHING BISIMULATION RELATION

In this section we show that for any security protocol $\Pi$, $T_\Pi$ and $\mathfrak{T}_\Pi$ are branching bisimilar modulo $\mathsf{Snd}$.

Let $\Pi = \{1, \cdots, n\}$ be the set of participants of a security protocol and consider $T_\Pi = (S, r, E)$ and $\mathfrak{T}_\Pi = (S', r', E')$. We recall that $S' \subseteq S$, $E' \subseteq E$ and $r' = r$, due to lemma 4. Define the binary relation $\leq \subseteq S \times S$ as $z \leq z'$ iff there is a path $z_0, \cdots, z_\ell$ in $T_\Pi$ with $\ell \geq 0$ such that $z_0 = z, z_\ell = z'$, and

- for all $0 \leq i < \ell$, there exists a $j \in \Pi$ where $z_i \overset{e}{\rightarrow} z_{i+1}$ with $e \in E_j$, $|en_j(z_i)| = 1$ and $en_j(z_i) \subseteq \mathsf{Snd}$.

We note that $\leq$ is a reflexive relation. The following lemma explains why the notation $\leq$ is chosen for this relation: If $z \leq z'$, then the set of $rcv$ events *available*

at $z$ is a subset of $rcv$ events available at $z'$. Here, we say $e$ is available at $z$ if $e \in en_i(z)$ for some $i \in \Pi$.

**Lemma 6.** *Consider $T_\Pi = (S, r, E)$ and $z, z' \in S$. Then $z \leq z' \implies \forall i \in \Pi.\ en_i(z) \cap \mathsf{Rcv} \subseteq en_i(z') \cap \mathsf{Rcv}$.*

*Proof:* The assumption $z \leq z'$ implies that there is a path $z_0, z_1, \cdots, z_\ell$ in $T_\Pi$ with $z_0 = z$, $z_\ell = z'$, and $z_j \xrightarrow{e_j} z_{j+1}$ for $0 \leq j < \ell$. We define $\kappa = \{k \in \Pi \mid \exists j.\ 0 \leq j < \ell \wedge k = [e_j]\}$. Now consider an arbitrary $i \in \Pi$. If $i \notin \kappa$ then due to the independence property $en_i(z) = en_i(z')$. If $i \in \kappa$, then choose the minimal $0 \leq j < \ell$ such that $[e_j] = i$. As $j$ is minimal, we have $en_i(z) = en_i(z_j)$ due to the independence property. By the definition of $\leq$, $en_i(z_j) \cap \mathsf{Rcv} = \emptyset$. Therefore, $en_i(z) \cap \mathsf{Rcv} = \emptyset$. This completes the proof. ∎

**Theorem 1.** *Let $\Pi$ be the set of participants of a security protocol. Then $T_\Pi$ and $\mathfrak{T}_\Pi$ are branching bisimilar modulo* Snd.

*Proof:* Let $T_\Pi = (S, r, E)$ and $\mathfrak{T}_\Pi = (S', r', E')$. We define the relation $R \subseteq S \times S'$ as $(z, z') \in R$ iff $z \leq z'$. We show that $R$ is a branching bisimulation relation modulo Snd. Below, as a convention, primed symbols refer to states in $\mathfrak{T}_\Pi$.

We first note that $(r, r') \in R$. Now assume $(z_1, z_1') \in R$ and $z_1 \xrightarrow{e} z_2$ in $T_\Pi$. The assumption $(z_1, z_1') \in R$ implies that there exists a path $y_0, \cdots, y_\ell$ such that $y_0 = z_1$, $y_\ell = z_1'$, $l \geq 0$ and $y_i \xrightarrow{e_i} y_{i+1}$ for $0 \leq i < \ell$ where $|en_{[e_i]}(y_i)| = 1$ and $en_{[e_i]}(y_i) \subseteq \mathsf{Snd}$. We distinguish two cases:

1) $[e] \neq [e_i]$ for all $0 \leq i < \ell$. Then due to the confluence property there exists a $z_2'$ such that $z_1' \xrightarrow{e} z_2'$ and $z_2 \xrightarrow{e_0 \cdots e_\ell} z_2'$.
   We claim $z_2 \leq z_2'$. Clearly there exists a path $x_0, \cdots, x_\ell$ such that $x_0 = z_2$, $x_\ell = z_2'$, $l \geq 0$ and $x_i \xrightarrow{e_i} x_{i+1}$ for $0 \leq i < \ell$. It remains to show that $|en_{[e_i]}(x_i)| = 1$ and $en_{[e_i]}(x_i) \subseteq \mathsf{Snd}$. Using the confluence property on the intermediate states we observe that for all $0 \leq i \leq \ell$, $y_i \xrightarrow{e} x_i$. Now, due to the independence property, $en_j(y_i) = en_j(x_i)$ for all $j \neq [e]$. That is, from $z_1 \leq z_1'$ we can conclude $z_2 \leq z_2'$.
2) $[e] = [e_j]$ for some $0 \leq j < \ell$. Let $j$ be the smallest number for which $[e] = [e_j]$. From $z_1 \leq z_1'$ we have $en_{[e]}(y_j) \subseteq \mathsf{Snd}$. Therefore $e_j \in \mathsf{Snd}$. We claim $e_j = e$. This is because $en_{[e]}(y_k) = en_{[e]}(z_1)$ for $k \leq j$, due to the independence property. Moreover $e \in en_{[e]}(z_1)$ and $|en_{[e]}(y_j)| = 1$ due to $z_1 \leq z_1'$. Therefore $e = e_j$. This results in $z_2 \xrightarrow{e_1 \cdots e_{j-1}} y_{j+1}$ due to the confluence property. Then, obviously, $z_2 \xrightarrow{e_1 \cdots e_{j-1} e_{j+1} \cdots e_{\ell-1}} z_1'$. Now, $z_1 \leq z_1'$ implies that $z_2 \leq z_1'$ and indeed $(z_1, z_2) \in \mathsf{Snd}$.

Now we consider the other direction: Assume $(z_1, z_1') \in R$ and $z_1' \xrightarrow{e'} z_2'$ in $\mathfrak{T}_\Pi$. The assumption $(z_1, z_1') \in R$ implies that $z_1 \xrightarrow{e_1 \cdots e_\ell} z_1'$ where $l \geq 0$ and $e_i \in \mathsf{Snd}$. Take $z_2 = z_2'$, and note that $z_1 \xrightarrow{e_1 \cdots e_\ell e} z_2$ and indeed $z_1' \leq z_1'$ and $z_2 \leq z_2'$. This completes the proof. ∎

We give a simple example of a branching bisimulation relation between $T_\Pi$ and $\mathfrak{T}_\Pi$.

**Example 1.** *Let $\Pi = \{1, 2\}$ and consider $T_\Pi$ and $\mathfrak{T}_\Pi$ of figure 1, where $s_i$ denotes $snd_i(m)$ for some message $m$ and $i \in \Pi$, and $r_i$ and $r_i'$ denote $rcv_i(m)$ for some message $m$ and $i \in \Pi$.*

*The following relation $R \subseteq S \times S'$ is a branching bisimulation relation modulo* Snd.

$$R = \{(A, A'), (D, D'), (E, E'), (F, F'), (G, G'), (A, E'), (B, D'), (C, F')\}$$

*To verify this we observe that:*

$$\begin{aligned}
A \xrightarrow{s_1} B &\implies A' \xrightarrow{s_2 s_1} D', (B, D') \in R \\
A \xrightarrow{s_1} B &\implies E' \xrightarrow{s_2} D', (B, D') \in R \\
A \xrightarrow{r_1} C &\implies A' \xrightarrow{s_2 r_1} F', (C, F') \in R \\
A \xrightarrow{r_1} C &\implies E' \xrightarrow{r_1} F', (C, F') \in R \\
A \xrightarrow{s_2} E &\implies A' \xrightarrow{s_2} E', (E, E') \in R \\
A \xrightarrow{s_2} E &\implies E' \xrightarrow{\epsilon} E', (E, E') \in R \\
B \xrightarrow{s_2} D &\implies D' \xrightarrow{\epsilon} D', (D, D') \in R \\
C \xrightarrow{s_2} F &\implies F' \xrightarrow{\epsilon} F', (F, F') \in R \\
E \xrightarrow{s_1} D &\implies E' \xrightarrow{s_1} D', (D, D') \in R \\
E \xrightarrow{r_1} F &\implies E' \xrightarrow{r_1} F', (F, F') \in R \\
E \xrightarrow{r_1'} G &\implies E' \xrightarrow{r_1'} G', (G, G') \in R
\end{aligned}$$

*Here $\epsilon$ is the empty sequence. Verifying the other direction of the relation is analogous. Note that $(z, z') \in R$ iff $z \leq z'$.*

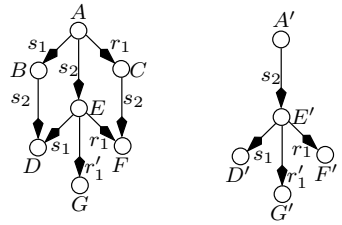


Figure 1. $T_\Pi$ (left) and $\mathfrak{T}_\Pi$ (right) of example 1.

## V. An ample subdag relation

In this section we show that for any security protocol $\Pi$, $\mathfrak{P}_\Pi$ is an ample subdag of $T_\Pi$ modulo Snd. This implies that these subdags are trace equivalent, modulo Snd.

The following lemma is adapted from well-known results in the partial order reduction literature, e.g. see [26, Theorem 4.2].

**Lemma 7.** *Let $T' = (S', r', E')$ be a subdag of $T = (S, r, E)$ with $W \subseteq E$, where the following three properties hold:*

- [C0] *For any state* $s \in S'$, $en^{T'}(s) = \emptyset$ *only if* $en^{T}(s) = \emptyset$.
- [C1] *Let* $s \xrightarrow{e_1 \cdots e_m} s'$ *in* $T$ *and assume* $en^{T'}(s) = \{\mathfrak{e}_1, \cdots, \mathfrak{e}_\ell\}$. *Then, for all* $1 \leq i \leq m$ *where* $[e_i] = [\mathfrak{e}_j]$ *for some* $j \in \{1, \cdots, \ell\}$, *we have* $\exists k \leq i.\ e_k \in en^{T'}(s)$.
- [C3] *For any state* $s \in S'$, $en^{T'}(s) \subset en^{T}(s)$ *implies that* $en^{T'}(s) \subseteq W$.

*Then* $T'$ *is an ample subdag of* $T$ *modulo* $W$. [1]

**Theorem 2.** *Let* $\Pi$ *be the set of participants of a security protocol. Then,* $\mathfrak{P}_\Pi$ *is an ample subdag of* $T_\Pi$ *modulo* Snd.

*Proof:* The proof goes by showing that $\mathfrak{P}_\Pi$ is a subdag of $T_\Pi$ and conditions C0,C1 and C3 of lemma 7 are satisfied. According to lemma 5, dag $\mathfrak{P}_\Pi$ is indeed a subdag of $T_\Pi$ and also condition C0 is satisfied. Condition C3 also clearly holds by the definition of $\mathfrak{P}_\Pi$. Below, it is shown that condition C1 is satisfied as well.

Let $s \xrightarrow{e_1 \cdots e_m} s'$ in $T$, $en^{T'}(s) = \{\mathfrak{e}_1, \cdots, \mathfrak{e}_\ell\}$, and $[e_i] = [\mathfrak{e}_j]$ for some $1 \leq i \leq m$ and $1 \leq j \leq \ell$. We claim $\exists k \leq i.\ e_k \in en^{T'}(s)$. By definition of $\mathfrak{P}_\Pi$, either $[\mathfrak{e}_1] = \cdots = [\mathfrak{e}_\ell]$ and $en^{T'}(s) \subseteq$ Snd, or $en^{T'}(s) = en^{T}(s)$. The claim clearly is true in the latter case. Therefore, below we focus on the former case and write $\lambda = [\mathfrak{e}_1]$.

We have $[e_i] = \lambda$ for some $1 \leq i \leq m$ by the assumption. Two cases are possible:

- If $i = 1$: By definition of $\mathfrak{P}_\Pi$, $[e_1] = \lambda$ iff $e_1 \in en^{T'}(s)$. We let $k = 1$ in this case.
- If $i > 1$: We assume for all $1 \leq k < i.\ e_k \notin en^{T'}(s)$ and prove $e_i \in en^{T'}(s)$. Let us write $s \xrightarrow{e_1 \cdots e_i} \hat{s}$. By definition of $\mathfrak{P}_\Pi$, we get $[e_k] \neq \lambda$, for all $1 \leq k < i$. By the confluence property, therefore, $en^{T}_{\lambda}(\hat{s}) = en^{T}_{\lambda}(s)$. Since $e_i \in en^{T}_{\lambda}(\hat{s})$, we conclude $e_i \in en^{T}_{\lambda}(s)$. Now, from $[e_i] = \lambda$ and the definition of $\mathfrak{P}_\Pi$ we get $e_i \in \{\mathfrak{e}_1, \cdots, \mathfrak{e}_\ell\}$; that is $e_i \in en^{T'}(s)$.

This completes the proof. ■

We give a simple example of an ample subdag relation between $T_\Pi$ and $\mathfrak{P}_\Pi$.

**Example 2.** *Consider figure 1. Note that the dag on the right follows the construction of pruned execution dags (section III-C). One can easily check that the dag on the right is an ample subdag of* $T_\Pi$ *(left) modulo* Snd. *Below, we verify the conditions C0, C1 and C3 (of lemma 7) for this dag.*

*It is obvious that the dag on the right satisfies conditions C0 and C3. Now, observe that the only sequences of events in* $T_\Pi$ *(left) which are not in the dag on the right are* $s_1 s_2$ *and* $r_1 s_2$, *both rooted at* $A$. *Since the only enabled event at* $A'$ *is* $s_2$, *and* $[s_2] \neq [s_1]$ *and* $[s_2] \neq [r_1]$, *we conclude that condition C1 is also met by the dag on the right.*

[1] Condition C2 that is omitted here refers to cycles in state spaces, which is immaterial for dags of finite depth.

## VI. POR ALGORITHMS

Here, we follow the *ample set* approach [26]. For a general introduction to POR, see [27]. A POR algorithm prescribes a method to select $ample(s)$ at each state $s$, with $ample(s) \subseteq en(s)$, such that exploring only elements of $ample(s)$, instead of the entire $en(s)$, is sufficient for generating a smaller execution dag which is "equivalent" to the execution dag (of a protocol). Only the equivalence relations given in definitions 1 and 2 (see section II) are considered here: ample subdag and branching bisimulation relations.

**Algorithm 1** $\mathbf{Gen}(\mathcal{A})$: Generates execution dags for branching security protocols

```
1:  Open := ∅; Closed := ∅
2:  Open.insert(r)
3:  while Open ≠ ∅ do
4:     s := Open.extract
5:     Closed.insert(s)
6:     if A = F then
7:        J := ∅
8:     if A = R then
9:        J := {j ∈ Π | |en_j(s)| = 1 ∧ en_j(s) ⊆ Snd}
10:    if A = P then
11:       J := {j ∈ Π | |en_j(s)| ≥ 1 ∧ en_j(s) ⊆ Snd}
12:    if J ≠ ∅ then
13:       I := {i | i = min J}
14:    else
15:       I := Π
16:    ample(s) := {e ∈ en(s) | [e] ∈ I}
17:    for all e ∈ ample(s) do
18:       if e(s) ∉ Closed ∧ e(s) ∉ Open then
19:          Open.insert(e(s))
```

$\mathbf{Gen}(\mathcal{A})$ specified in algorithm 1 implements three different methods for generating execution dags. The value of parameter $\mathcal{A}$ determines the execution model: (1) $F$ indicates the full execution dag of section III-A, (2) $R$ indicates the reduced execution dag of section III-B, and (3) $P$ indicates the pruned execution dag of section III-C. The input to algorithm 1 is a (symbolic) specification of $T_i$ for all $i \in \Pi$ and the attacker's initial knowledge $K_0$. These in particular determine the initial state of the execution dag, which is as usual denoted by $r$ in the algorithm. The algorithm explores the indicated execution dag of the protocol; here, we confine to the traversal strategy and abstract away from generating the output (file).

The set $Open$ contains visited, but not yet expanded states, and $Closed$ is the set of visited and expanded states. When $Open$ is implemented as a queue, the resulting traversal strategy is breadth-first, while implementing $Open$ as a stack results in a depth-first strategy. We remark that the purpose of using the $Closed$ set in algorithm 1 is to avoid state

revisits. This set is not needed to guarantee the termination of the algorithm, because dags are acyclic. Therefore, this set can gradually be removed from memory and be stored on high latency media (e.g. disks), in case memory limits are reached, without endangering the termination of the algorithm. For similar approaches to memory management see [20]. We remark that the algorithm can be extended in the obvious way to allow on-the-fly checks for reachability security properties.

The following theorem is immediate by theorems 1 and 2.

**Theorem 3.** *Let* $\Pi$ *be the set of participants of a security protocol. Then:*

- $\mathbf{Gen}(R)$ *is branching bisimilar to* $\mathbf{Gen}(F)$ *modulo* Snd.
- $\mathbf{Gen}(P)$ *is an ample subdag of* $\mathbf{Gen}(F)$ *modulo* Snd.

### A. Remarks

The algorithm of CJM simply peaks a $snd$ event if any is enabled at state $s$, and considers the selected action as the set $ample(s)$. If no such event is enabled, then it lets $ample(s) = en(s)$. It is easy to observe that in protocols with branching participants this algorithm does not create an ample subdag of the (full) execution dag.

**Example 3.** *Consider the following two processes:* $p = snd_p(m_1) + snd_p(m_2)$ *and* $q = rcv_q(m_1) + rcv_q(m_2)$*, with* '$+$' *being the* choice *operator (thus corresponds to a fork in the dag describing these processes), and* $m_1$ *and* $m_2$ *being two messages such that* $m_2 \notin \mathfrak{C}\{m_1\}$*. Consider an execution of these processes in presence of an attacker with empty initial knowledge. Obviously if only one of the two alternative* $snd_p$ *events are explored in a POR algorithm, the resulting state space would not include the other* $snd_p$ *event. That is, the state space resulting from such a POR algorithm would not be an ample subdag of the full state space which contains both* $rcv_q(m_1)$ *and* $rcv_q(m_2)$ *events.*

Note that internal actions are allowed in $\mathbf{Gen}(\mathcal{A})$, and these will be treated as $rcv$ events by the algorithm (see also section III). Note that in theorem 3 only $snd$ events are silent. That is, $rcv$ events and internal actions can be referred to in properties that are preserved by branching bisimulation (or, ample subdag equivalence).

If the attacker needs to perform an internal action, e.g. to signal out an event which is interesting for verification, then a *dummy* process can be used that receives a designated message from the attacker and performs the corresponding internal action. Therefore, it is not necessary to provide the attacker with explicit internal actions.

## VII. A CASE STUDY

Below, in the context of a case study, we evaluate the effectiveness of the proposed POR algorithms in terms of the number of states and transitions they explore, and their execution time. Our experiments are performed on a finite-state approximation of an optimistic fair exchange protocol for trusted computing modules, described in [32]. The finite-state nature of the case study enables us to compare the concrete number of explored states and transitions. Naturally the proposed algorithms are not bound to finite-state settings. For instance, $\mathbf{Gen}(R)$ can be used with the constraint solving algorithm of [21] for verifying fairness of contract signing protocols.

The protocol of [32] assumes a finite set of trusted computing modules $C$, with each $c \in C$ having access to a set of nonces $N_c$, and a finite set of trusted entities $T$. In case of unexpected interruptions, wronged modules can resort to one of the trusted entities. The goal of the protocol is to secure fair exchange of digital items among the members of $C$, in the presence of malicious players.

We have specified the protocol in the $\mu$CRL process algebraic language [19]. The $\mu$CRL toolset [7] version 2.18.5 is extended with algorithms $\mathbf{Gen}(R)$ and $\mathbf{Gen}(P)$, and used to generate state spaces corresponding to various instantiations of the protocol. The state space generation algorithms of the toolset already included $\mathbf{Gen}(F)$. The extended $\mu$CRL toolset, along with the specifications of the case studies reported here, can be found at http://www.win.tue.nl/~awijs/spor/securitymcrl.html.

We remark that the $\mu$CRL toolset provides a general-purpose state space generation framework, and is not tailored for security applications. The generation times reported here can therefore be improved using special purpose model checkers, such as those developed in AVISPA [3]. This observation is however immaterial in assessing the effectiveness of POR algorithms, as the amount of *reduction* in time and memory usage is generally independent to the underlying technology.

In our experiments the variables are: (1) the size of set $T$, which intuitively amounts to the number of parallel processes in the system, and (2) the size of set $N_c$, which roughly corresponds to the number of sequential transactions each element of $C$ can perform. The size of set $C$ is fixed to two in all the experiments: It takes at least two modules to execute the fair exchange protocol.

A security requirement for the protocol of [32] is *fairness*. Suppose modules $c_1$ and $c_2$ possess digital items $d_1$ and $d_2$, respectively, and wish to exchange these items. Fairness asserts that if $c_1$ receives $d_2$, then $c_2$ has always a way to receive $d_1$. In $\mathrm{CTL}^*_{-\mathrm{X}}$ this can be formalized as

$$\mathbf{AG}(recv(c_1, d_2) \implies \mathbf{EF} recv(c_2, d_1))$$

where $\mathbf{A}, \mathbf{E}, \mathbf{G}$ and $\mathbf{F}$ have their standard meaning [9]. Note that once $recv(c_1, d_2)$ becomes true in a state, it will persistently remain true in all the subsequent states. The property is indeed preserved under branching bisimilarity modulo Snd events.

*Experiment setting:* All experiments were performed on a machine with two dual-core AMD OPTERON (tm) processors 885 2.6 GHz, 126 GB RAM, running RED HAT 4.3.2-7. The $\mu$CRL toolset supports distributed state space generation [6], allowing multiple CPUs, either on different machines or on a single machine, to generate a state space in collaboration. We implemented the algorithms in this paper for a distributed setting, as this can be done in a straightforward fashion (the cycle proviso C2 of POR is immaterial in the setting of security protocols). Therefore, the results presented here are obtained by running two exploration processes simultaneously on a machine. [2] Table I reports our results. Measured time refers to wall clock time, and is expressed as '(hours):(minutes):(seconds)'.

Compared to $\mathbf{Gen}(R)$, in general $\mathbf{Gen}(P)$ allows more reduction, because of its more relaxed constraints on the ample sets. However, in this case study, $\mathbf{Gen}(P)$ and $\mathbf{Gen}(R)$ have little differences. This is due to the structure of the protocol, where processes can do at most one send event at branching points. This is a common feature of the class of optimistic fair exchange protocols, cf. [4].

By increasing $T$ the effectiveness of $\mathbf{Gen}(P)$ and $\mathbf{Gen}(R)$ is increased more and more compared to $\mathbf{Gen}(F)$, while an increase in the number of nonces $|N_c|$ has a smaller effect on the amount of reduction. This is because $T$ directly relates to the number of parallel processes, and the effectiveness of POR in general favors larger number of parallel processes.

In table I, we observe that the number of explored states is reduced up to $30\%$ using POR; the number of explored transitions is on average $59\%$ reduced, while the amount of reduction in exploration time is up to $37\%$. Although POR loads the generation algorithm with some book-keeping and extra computations, the overall reduction gained definitely compensates for these costs, as is evident in the experimental results. It is worth mentioning that next-state generation is in general relatively slow in security protocols, as it involves matching the messages that protocol participants can receive with the messages that the attacker process can construct.

## VIII. RELATED WORK

Algorithms for POR are an established branch of model checking and state space generation research, see, e.g., [27]. As finding an optimal $ample$ set is NP-hard [25], many POR algorithms in the literature focus on a particular setting and propose heuristics for computing a (near-)optimal $ample$ set in that setting, e.g. see [26]. Our work can be seen as one of these heuristics for a large class of security protocols.

POR techniques for analyzing security protocols can be traced back to the work of Shmatikov and Stern [30], where some methods to reduce the number of states when verifying security protocols are proposed, but no formalization of the techniques is provided. Clarke Jha and Marrero [10] were the first to present a POR algorithm for security protocols and formally prove its properties. Below, we give a chronological overview on major existing POR(-like) reduction techniques for the verification of security protocols. These techniques *all* aim at linear-time properties, whereas we have considered branching-time security requirements besides linear-time properties. This feature distinguishes our work from previously known results. The existing techniques mostly consider protocols with non-branching participants, while the focus of our work is on accommodating branching participants.

The algorithm of CJM is tailored for non-branching security protocols [10]. The logic that is used in [10] is a variant of past-time LTL extended with an operator $\mathcal{E}$ for explicit reference to the knowledge of participants and the attacker. For instance, they can specify properties like $\mathcal{E}_{\mathrm{DY}}(m) \implies \neg\exists p \in \Pi.\ \Diamond^{-1}\mathcal{E}_p(m)$, where $m \in Msg$ and $\mathcal{E}_x(m)$ states that the process $x$ (in the aforementioned formula the Dolev-Yao attacker DY) *knows* $m$, that is $m \in \mathfrak{C}(K_x)$ with $K_x$ being the knowledge set of process $x$. Here, given a trace $\alpha = \alpha_1 \cdots \alpha_i \cdots$, let $\alpha^{i\infty} \models \Diamond^{-1}\phi$ iff $\exists j.\ 1 \leq j \leq i \wedge \alpha^{j\infty} \models \phi$. The aforementioned property thus intuitively stipulates that the knowledge of honest processes is never shared with the attacker. In order for their POR algorithm to preserve this logic, it is required that terms which refer to the attacker's knowledge appear only negatively in the properties. To see why this constraint is needed, consider the property $\phi = (\mathcal{E}_p(t_0) \implies \Diamond^{-1}\mathcal{E}_{\mathrm{DY}}(t_1))$, stating that if the honest process $p$ knows $t_0$, then in some past time the attacker knew $t_1$. Now, consider processes $p = rcv_p(t_0).\mathsf{stop}$, and $q = snd_q(t_1).\mathsf{stop}$ such that $t_0 \in \mathfrak{C}(K_0)$ with $K_0$ being attacker's initial knowledge, but $t_1 \notin \mathfrak{C}(K_0)$. If $snd_q$ is prioritized over $rcv_p$, then property $\phi$ would hold in the reduced state space, while $\phi$ is violated in the full state space.

Avoiding positive reference to the attacker's knowledge is tightly related to the novelty of the POR algorithm of CJM for security protocols: Roughly speaking, standard POR algorithms for Kripke structures rely on the notion of *invisible* actions. An action is called invisible if $s \xrightarrow{\alpha} s'$ implies that the same set of propositions hold at $s$ and $s'$. Propositions which are relevant for security protocols are the knowledge of the attacker and the knowledge of the participants. Observe that $snd$ events (potentially) change the knowledge of the attacker, and $rcv$ events (potentially) change the knowledge of the participants. Therefore, strictly speaking, there is no invisible action in a security protocol; hence standard POR algorithms for communication protocols (e.g. as implemented in SPIN [9]) are not effective for security protocols. The novelty of CJM's algorithm is in exploiting the fact that $snd$ events change the knowledge of the attacker only in the positive way, namely the

[2] A comparison between algorithms $\mathbf{Gen}(F)$ and $\mathbf{Gen}(P)$ using the distributed $\mu$CRL toolset running on a cluster of machines can be found in [33].

Table I
EXPERIMENTAL COMPARISON OF $\mathbf{Gen}(F)$, $\mathbf{Gen}(P)$ AND $\mathbf{Gen}(R)$

| Instance | | $\mathbf{Gen}(F)$ | | | | $\mathbf{Gen}(P)$ & $\mathbf{Gen}(R)$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | State space size | | $\mathbf{Gen}(P)$ | | $\mathbf{Gen}(R)$ | |
| $\lvert N_c \rvert$ | $T$ | #States | #Trans. | Mem. use | Time | #States | #Trans. | Mem. use | Time | Mem. use | Time |
| 1 | 2 | 280,962 | 1,212,928 | 0.09 GB | 00:04:26 | 207,028 | 498,560 | 0.09 GB | 00:03:42 | 0.09 GB | 00:03:37 |
| 3 | 2 | 1,760,306 | 6,434,728 | 0.45 GB | 00:44:26 | 1,375,384 | 3,040,670 | 0.36 GB | 00:36:42 | 0.40 GB | 00:36:27 |
| 5 | 2 | 33,179,272 | 110,605,296 | 8.05 GB | 32:11:57 | 27,151,416 | 58,672,398 | 7.13 GB | 26:05:54 | 7.23 GB | 25:50:47 |
| 1 | 3 | 13,246,976 | 71,570,596 | 2.59 GB | 04:36:48 | 8,844,384 | 24,087,744 | 2.40 GB | 03:35:35 | 2.48 GB | 03:17:45 |
| 2 | 3 | 46,965,265 | 238,641,068 | 9.80 GB | 22:10:09 | 31,159,497 | 80,560,109 | 8.42 GB | 15:54:48 | 8.77 GB | 15:20:39 |
| 3 | 3 | 176,477,749 | 814,529,294 | 28.95 GB | 127:48:31 | 123,540,939 | 308,440,601 | 26.81 GB | 93:29:09 | 24.59 GB | 81:11:56 |

knowledge of the attacker is not decreased. Therefore, if the attacker's knowledge is only referred negatively, then $snd$ events become *semi*-invisible [10]. In our formalization, this observation is translated into: $snd$ events are inert silent actions, i.e. internal transitions between branching bisimilar states.

Basin, Mödersheim and Viganò [5] introduce *constraint differentiation* as a means to effectively integrate POR-like techniques into the lazy attacker setting (i.e. symbolics representation of message exchanges, plus a suitable set of constraints to encode the attacker's abilities). They state that merely prioritizing some actions over others in order to avoid independent interleavings does not gain sufficient reduction in the lazy attacker framework. Therefore, their focus is to detect, and to avoid re-analyzing, overlapping constraints appearing in different interleavings. The constraint differentiation method also subsumes the POR algorithm of Cremers and Mauw [14].

Escobar, Meadows and Meseguer give a collection of state-space reduction techniques for analyzing security protocols in Maude-NRL [16]. The POR algorithm proposed in [16] is the counterpart of CJM's algorithm in backward search.

## IX. CONCLUDING REMARKS

Two extensions of the POR algorithm of Clarke, Jha and Marrero have been presented. The proposed algorithms are suitable for branching security protocols, e.g. optimistic fair contract signing schemes. The first extension is proved to generate a reduced state space which is branching bisimilar to the full state space, while the second extension generates a state space that is trace equivalent to the full state space. Branching bisimilarity, while ignoring silent actions, preserves the branching structure of the state space. Therefore, branching-time security properties, e.g. expressed in $\mathrm{CTL}^*_{-\mathrm{X}}$, are preserved under the first POR algorithm. The second algorithm is suitable when the security requirements of the system pertain to linear-time properties, e.g. expressed in $\mathrm{LTL}_{-\mathrm{X}}$. The second POR algorithm can in general achieve more reductions compared to the first POR algorithm, due to its relaxed conditions for constructing *ample* sets.

Experimental results using an implementation of the algorithms in the toolset of the $\mu$CRL process algebra are reported. These preliminary experiments indicate considerable reduction in the number of explored states and transitions, and overall time required for exploration when using the proposed POR algorithms. Further experiments with various classes of security protocols, and implementing the POR algorithms into a constraint solver for verifying security protocols (such as [21]), is left for future work.

We believe the branching bisimulation relation established in this paper can be of independent interest, as it formally connects infinitely branching state spaces (of finite depth) for two execution models of security protocol. We intend to investigate extending this bisimulation relation to protocol instantiations with an unbounded number of sessions. The extension is nontrivial because, contrary to our current settings, state spaces of such instantiations would not be of finite depth.

POR algorithms which preserve the alternating-time temporal logic [2] are another interesting topic for future research. Indeed many security properties for contract signing protocols are expressible as game-based strategies, which can be formalized in the alternating-time temporal logic [22].

## ACKNOWLEDGEMENTS

We are grateful to Stefan Leue for helpful discussions. Mohammad Torabi Dashti has been partially supported by AVANTSSAR, FP7-ICT-2007-1 Project no. 216471.

## REFERENCES

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01*, pages 104–115, 2001.

[2] R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *FOCS '97*, pages 100–109. IEEE CS, 1997.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Hankes Drielsma, P. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV '05*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[4] N. Asokan. *Fairness in Electronic Commerce*. PhD thesis, University of Waterloo, 1998.

[5] D. Basin, S Mödersheim, and L Viganò. CDiff: A new reduction technique for constraint-based analysis of security protocols. In *CCS '03*, pages 335–344. ACM Press, 2003.

[6] S. Blom, J. Calamé, B. Lisser, S. Orzan, J. Pang, J. van de Pol, M. Torabi Dashti, and A. Wijs. Distributed analysis with $\mu$CRL: A compendium of case studies. In *TACAS '07*, volume 4424 of *LNCS*, pages 683–689. Springer, 2007.

[7] S. Blom, W. Fokkink, J. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. $\mu$CRL: A toolset for analysing algebraic specifications. In *CAV '01*, volume 2102 of *LNCS*, pages 250–254, 2001.

[8] T. Chothia, S. Orzan, J. Pang, and M. Torabi Dashti. A framework for automatically checking anonymity with $\mu$CRL. In *TGC '06*, volume 4661 of *LNCS*, pages 301–318. Springer, 2006.

[9] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[10] E. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *TACAS '00*, volume 1785 of *LNCS*, pages 503–518. Springer, 2000.

[11] E. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4):443–487, 2000.

[12] E. Clarke, S. Jha, and W. Marrero. Efficient verification of security protocols using partial-order reductions. *STTT*, 4(2):173–188, 2003.

[13] M. Clarkson and F. Schneider. Hyperproperties. In *CSF '08*, pages 51–65. IEEE Computer Society, 2008.

[14] C. Cremers and S. Mauw. Checking secrecy by means of partial order reduction. In *SAM '04*, volume 3319 of *LNCS*, pages 177–194. Springer, 2005.

[15] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, IT-29(2):198–208, 1983.

[16] S. Escobar, C. Meadows, and J. Meseguer. State space reduction in the Maude-NRL protocol analyzer. In *ESORICS '08*, volume 5283 of *LNCS*, pages 548–562. Springer, 2008.

[17] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Inf. Comput.*, 150(2):132–152, 1999.

[18] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

[19] J. Groote and A. Ponse. The syntax and semantics of $\mu$CRL. In *Algebra of Communicating Processes '94*, pages 26–62. Springer, 1995.

[20] M. Hammer and M. Weber. "To store or not to store" reloaded: Reclaiming memory on demand. In *FMICS '06*, volume 4346 of *LNCS*, pages 51–66. Springer, 2006.

[21] D. Kähler and R. Küsters. Constraint solving for contract-signing protocols. In *CONCUR '05*, volume 3653 of *LNCS*, pages 233–247. Springer, 2005.

[22] S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange. In *CONCUR '01*, volume 2154 of *LNCS*, pages 551–565. Springer, 2001.

[23] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS '01*, pages 166–175. ACM Press, 2001.

[24] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458–487, 1995.

[25] D. Peled. All from one, one for all: On model checking using representatives. In *CAV '93*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.

[26] D. Peled. Partial order reduction: Linear and branching temporal logics and process algebras. In Peled et al. [27], pages 233–257.

[27] D. Peled, V. Pratt, and G. Holzmann, editors. *Partial Order Methods in Verification*, volume 29 of *DIMACS*. AMS Press, 1997.

[28] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI '05*, pages 1252–1259, 2005.

[29] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *ICAPS '05*, pages 2–11. AAAI, 2005.

[30] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *CSFW '98*, pages 106–115. IEEE CS, 1998.

[31] F.J. Thayer Fabrega, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *J. Comput. Secur.*, 7(2-3):191–230, 1999.

[32] M. Torabi Dashti, S. K. Nair, and H. Jonker. Nuovo DRM Paradiso: Designing a secure, verified, fair exchange DRM scheme. *Fundam. Inform.*, 89(4):393–417, 2008.

[33] M. Torabi Dashti, A. Wijs, and B. Lisser. Distributed partial order reduction for security protocols. In *PDMC '07*, volume 198 of *ENTCS*, pages 92–99, 2008.