2017 IEEE European Symposium on Security and Privacy

A Novel Approach for Reasoning about Liveness in Cryptographic Protocols and its Application to Fair Exchange

Michael Backes[*‡], Jannik Dreier[†], Steve Kremer[†] and Robert Künnemann[*]
[*]*CISPA, Saarland University*
Saarland Informatics Campus
[†]*LORIA, Inria Nancy - Grand Est & CNRS & Université de Lorraine*
[‡]*MPI-SWS*
Saarland Informatics Campus

***Abstract*—In this paper, we provide the first methodology for reasoning about liveness properties of cryptographic protocols in a machine-assisted manner without imposing any artificial, finite bounds on the protocols and execution models. To this end, we design an extension of the SAPiC process calculus so that it supports key concepts for stating and reasoning about liveness properties, along with a corresponding translation into the formalism of multiset rewriting that the state-of-the-art theorem prover Tamarin relies upon. We prove that this translation is sound and complete and can thereby automatically generate sound Tamarin specifications and automate the protocol analysis.**

Second, we applied our methodology to two widely investigated fair exchange protocols – ASW and GJM – and to the Secure Conversation Protocol standard for industrial control systems, deployed by major players such as Siemens, SAP and ABB. For the fair exchange protocols, we not only re-discovered known attacks, but also uncovered novel attacks that previous analyses based on finite models and a restricted number of sessions did not detect. We suggest fixed versions of these protocols for which we prove both fairness and timeliness, yielding the first automated proofs for fair exchange protocols that rely on a general model without restricting the number of sessions and message size. For the Secure Conversation Protocol, we prove several strong security properties that are vital for the safety of industrial systems, in particular that all messages (e.g., commands) are eventually delivered in order.

1. Introduction

The security properties of cryptographic protocols are commonly formalized in terms of trace properties (*"all protocol traces are secure"*) or in terms of indistinguishability (*"the adversary cannot distinguish two protocol executions"*). Trace properties can be further partitioned into safety properties (*"bad things do not happen"*) and liveness properties (*"eventually, good things happen"*) [23].

An impressive number of symbolic analysis tools have been developed for automated reasoning about trace properties such as authentication and weak forms of confidentiality [25, 2, 14, 13, 27], and, more recently, also about indistinguishability properties such as anonymity and strong forms of confidentiality [6, 5, 10]. Reasoning about the class of *liveness* properties of cryptographic protocols, has, however, received considerably less attention in the literature, even though this class is vital for many security-sensitive applications. Fair exchange protocols arguably constitute the most widely known of these applications. These protocols ensure that both participants eventually reach a state that is fair, e.g., either they both receive a desired item, or no one does [3]. Further examples comprise self-healing schemes that ensure that the system eventually returns to a safe state (e.g., by providing a revocation API that ensures that compromised keys will be turned invalid [12]) and the Secure Conversation Protocol (SCP) – a security layer for industrial control systems specified in the United Architecture (UA) standard [24] that strives to ensure that all messages will eventually be received in the correct order.

While a few previous attempts on the machine-assisted verification of liveness properties exist (see the section on related work for more details), they all face one of the following two conceptual limitations.

First, the bulk of these attempts are restricted to *finite models* and hence fail to adequately model the variety of interactions that might arise in a protocol execution in concurrent environments. So far, only a bounded (typically one or two) number of concurrent sessions was considered, sometimes with an additional bound on the size of messages. Therefore, these approaches may miss common attacks that rely on interweaving different sessions, and even more so in cases of bounded message size.

Second, the only automated security protocol verification tool that is capable of expressing liveness properties without imposing such finite bounds onto the model is the Tamarin prover [26, 27]. The protocol description language of Tamarin relies on multiset rewriting, which constitutes a rich formalism for expressing desired protocols but offers a low level of abstraction. For instance, there is no abstract notion of a local computation, of individual protocol parties, or of the communication between two parties. This lack of a suitable abstraction layer makes it cumbersome and error-prone to express common assumptions imposed on virtually all existing protocols: certain messages will be

© 2017, Michael Backes. Under license to IEEE.
DOI 10.1109/EuroSP.2017.12

IEEE computer society

eventually delivered; honest protocol parties do not stall computation; and the existence of a mechanism that prevents protocol participants from waiting indefinitely for message delivery. Stating these assumptions is a key requirement for liveness properties in practice. Using an illustrative analogy, this is akin to stating a condition on, and reasoning about, objects within assembly code, compared to using a higher-level language instead. A potential remedy to overcome these limitations would be to provide a sound and complete embedding from a suitable higher-level language into the formalism of multiset rewriting. This would in particular leverage the impressive potential of Tamarin. However, this task contains formidable research challenges and no such prior work exists.

1.1. Our Contribution

This paper makes the following two main contributions: (i) the first methodology for reasoning about liveness properties in a machine-assisted manner without imposing any finite bounds on the models and (ii) an application of this methodology for verifying liveness properties in fair exchange protocols and in a standardized communication protocol for industrial control systems, discovering novel attacks and proving fixed versions secure.

Methodology for reasoning about liveness properties. We present the first verification toolchain for cryptographic protocols that can handle liveness properties such as fairness without the need to bound the number of sessions or the message size. To this end, we have designed an extension of the SAPiC process calculus [18, 19] such that it supports key elements for stating and reasoning about liveness, see below, along with a corresponding translation into multiset rewriting as used by Tamarin. We prove the correctness of the translation, retaining the completeness and soundness of both SAPiC and Tamarin. We can only provide a glimpse on this correctness proof in this paper due to space limitations, but encourage the reader to inspect the full-fledged, 30-page proof in the full version [4].

Our extension of SAPiC in particular supports three important concepts: *non-deterministic choice*, *local progress* and *resilient channels*. First, extending the calculus with (external) non-deterministic choice (NDC) is essential to model scenarios in which a participant needs to either continue the main protocol or execute one of the sub-protocols. As a technical complication, we stress that, unlike internal NDC, external NDC cannot be encoded using private channels. Second, local progress ensures that honest participants execute their protocol as far as possible, which differs from executions in traditional symbolic models. This is a key aspect for reasoning about liveness properties of the form "on all traces we eventually reach a state such that ..." in a meaningful way, as we need to discard partial traces where the participant would merely stop. Finally, resilient channels ensure that a message will eventually be delivered, but may be delayed by the attacker for an arbitrary amount of time. For fair exchange protocols, the impossibility of achieving fair exchange without a TTP [15] implies that a reliable channels between the protocol participants and the trusted third party (TTP) are strictly necessary.

We have implemented all our extensions in the SAPiC/-Tamarin toolchain [18, 19]. The undecidability of the underlying problem ensures that we cannot expect guaranteed termination of our tool. However, the underlying Tamarin prover allows to switch to interactive mode or to additionally specify lemmas, which are proved automatically, and may guide the tool. Interestingly, in our case studies, only one such additional lemma was needed and all our proofs are fully automatic. Implementing local progress turned out to be surprisingly involved, as it interacts with branching and operators for NDC that are possibly nested. Given a position in the current process, the next position the process needs to progress to is neither unique nor is there a dedicated set of next positions: instead, a propositional formula describes which positions have to be reached. Fortunately, we are able to treat the original, comprehensive correctness proof for SAPiC [19] in a blackbox manner, making the argument more conveniently accessible and less prone to human error.

Application to liveness-sensitive protocols. We have used our methodology and tool for modelling and analyzing two widely investigated fair exchange / contract-signing protocols: ASW [3] and GJM [16]. Moreover, we investigated a toy example for motivating the need of the so-called timeliness property. Our analyses not only re-discovered a known attack on the ASW protocol, first found by Shmatikov and Mitchell [29], but it additionally discovered a thus far unknown variant of their attack. Whether these findings should be considered attacks has been subject to discussions in the literature already though, since they strongly depend on what precisely should be considered a formal contract. If one slightly relaxes the notion of what constitutes a contract, we are able to prove both the fairness and timeliness properties of these protocols. To the best of our knowledge, this yields the first automated proof of fair exchange protocols that considers a general model without restriction on the number of sessions and message size. We finally show that our methodology and tool is applicable beyond fair exchange protocols by investigating the Secure Conversation Protocol standard [24] for industrial control systems, employed, e.g., by Siemens, SAP and ABB. We formally prove several strong security properties for this protocol that are vital for the safety of industrial systems, pertaining to the content, order, and number of messages (e.g., commands) transmitted during communication.

1.2. Related work

Cederquist and Torabi Dashti [7] present a first symbolic model with support for liveness properties. Their model formalizes the *resilient communication channel* assumption, by means of a fairness[1] constraint: if a given message

1. Fairness in this context refers to the fairness property in temporal logic, not to the security property of fair exchange protocols.

delivery is enabled infinitely often, the message must eventually be delivered. However, they use the general purpose algebra μCRL which does not have built-in support for a symbolic, Dolev-Yao-style adversary; moreover, it does not provide tool support for infinite state systems which would be necessary to analyse protocols without bounding message size or the number of sessions.

Liveness properties arise naturally in the study of optimistic fair exchange protocols (see [20] for a survey) which have been subject to many attempts of formal analyses.

Optimistic fair exchange protocols have been analysed through complex hand proofs in general settings [8, 11, 28], but these proofs have not been machine-checked. In [29], Shmatikov and Mitchell analyse the ASW and GJM protocol in the finite-state model checker Murϕ [29], using an ad-hoc encoding of processes in terms of finite-state machines. As they use a finite-state tool their model requires to bound both the message size and the number of sessions. They define fairness as a state invariant that must hold in any final state. However, they do not verify any liveness property, such as timeliness. In their analysis, they discovered a potential attack related to the precise definition of what is a contract. In this work, we were able to find a similar attack on their "fixed" protocol, which was surprisingly overlooked in their model. Kremer et al. use another finite model checker, Mocha, to analyse several fair exchange [21], contract signing [22] and multi-party contract signing [9]. They model resilient channels as fairness constraints and are able to check liveness properties. Gürgens and Rudolph [17] use the finite-state model checker SHVT to find new flaws in some fair non-repudiation protocols (flaws that were outside of the model of previous analyses). This illustrates that finite model checkers may easily miss attacks, and that there is consequently a need for a general model that is capable of reasoning about liveness properties without imposing artificial finiteness constraints on the underlying model.

2. Preliminaries

Terms and equational theories. As usual in symbolic protocol analysis, we model messages by abstract terms. Therefore, we define an order-sorted term algebra with the sort msg and two incomparable subsorts pub and $fresh$. For each of these subsorts we assume a countably infinite set of names, FN for fresh names and PN for public names. Fresh names will be used to model cryptographic keys and nonces while public names model publicly known values. We, furthermore, assume a countably infinite set of variables for each sort s, $\mathcal{V}_s$, and let $\mathcal{V}$ be the union of the set of variables for all sorts. We write $u : s$ when the name or variable u is of sort s. Let Σ be a signature, i.e., a set of function symbols, each with an arity. We write f/n when function symbol f is of arity n. There is a subset $\Sigma_{priv} \subseteq \Sigma$ of *private* function symbols, which cannot be applied by the adversary. We denote by $\mathcal{T}_\Sigma$ the set of well-sorted terms built over Σ, PN, FN and $\mathcal{V}$. For a term t, we denote by $names(t)$, respectively $vars(t)$ the set of names, respectively variables, appearing in t. The set of ground terms, i.e., terms without variables, is denoted by $\mathcal{M}_\Sigma$. When Σ is fixed or clear from the context, we often omit it and simply write $\mathcal{T}$ for $\mathcal{T}_\Sigma$ and $\mathcal{M}$ for $\mathcal{M}_\Sigma$.

We equip the term algebra with an equational theory E, which is a finite set of equations of the form $M = N$ where $M, N \in \mathcal{T}$. From the equational theory we define the binary relation $=_E$ on terms, which is the smallest equivalence relation containing equations in E that is closed under application of function symbols, bijective renaming of names and substitution of variables by terms of the same sort. Furthermore, we require E to distinguish different fresh names, i. e., $\forall a, b \in FN : a \neq b \Rightarrow a \neq_E b$.

Example 1. *Digital signatures can be modelled using a signature*

$$\Sigma = \{\, sig/2, ver/2, pk/1, sk/1 \,\}$$

and an equational theory defined by

$$ver(sig(m, sk(i)), pk(i)) = m,$$

where i is the identity of a party. If sk is a private function symbol, this gives a very minimalistic model of a public-key infrastructure.

For the remainder of the article, we assume that E refers to some fixed equational theory and that the signature and equational theory always contain symbols and equations for pairing and projection, i.e., $\{\langle ., . \rangle, \mathsf{fst}, \mathsf{snd}\} \subseteq \Sigma$ and equations $\mathsf{fst}(\langle x, y \rangle) = x$ and $\mathsf{snd}(\langle x, y \rangle) = y$ are in E. We will sometimes use $\langle x_1, x_2, \ldots, x_n \rangle$ as a shortcut for $\langle x_1, \langle x_2, \langle \ldots, \langle x_{n-1}, x_n \rangle \ldots \rangle$.

Positions within terms are defined as usual. A position p is a sequence of positive integers and $t|_p$ denotes the subterm of t at position p.

Facts. We also assume an unsorted signature Σ_{fact}, disjoint from Σ. The set of *facts* is defined as

$$\mathcal{F} := \{F(t_1, \ldots, t_k) \mid t_i \in \mathcal{T}_\Sigma, F \in \Sigma_{fact} \text{ of arity } k\}.$$

Facts will be used both to annotate protocols by means of events and to define multiset rewrite rules. We partition the signature Σ_{fact} into *linear* and *persistent* fact symbols. We suppose that Σ_{fact} always contains a persistent, unary symbol !K and a linear, unary symbol Fr. Given a sequence or set of facts S we denote by $lfacts(S)$ the multiset of all linear facts in S and $pfacts(S)$ the set of all persistent facts in S. By notational convention, facts whose identifier starts with '!' will be persistent. $\mathcal{G}$ denotes the set of ground facts, i.e., the set of facts that does not contain variables. For a fact f we denote by $ginsts(f)$ the set of ground instances of f. This notation is also lifted to sequences and sets of facts as expected.

Predicates. We assume an unsorted signature Σ_{pred} of predicate symbols that is disjoint from Σ and Σ_{fact}. The set of *predicate formulas* is defined as

$$\mathcal{P} := \{pr(t_1, \ldots, t_k) \mid t_i \in \mathcal{T}_\Sigma, pr \in \Sigma_{pred} \text{ of arity } k\}.$$

Predicate formulas may be used to describe branching conditions in protocols. While our translation supports these predicates, none of our case studies requires any predicate except for checking equality, i.e., $equal/2 \in \Sigma_{pred}$ with $\phi_{equal}(x, y) = x \approx y$. We refer to the full version for details.

Substitutions. A substitution σ is a partial function from variables to terms. We suppose that substitutions are well-typed, i.e., they only map variables of sort s to terms of sort s, or of a subsort of s. We denote by $\sigma = \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$ the substitution whose domain is $\mathbf{D}(\sigma) = \{x_1, \ldots, x_n\}$ and which maps x_i to t_i. As usual, we homomorphically extend σ to apply to terms and facts, and use a postfix notation to denote its application, e.g., we write $t\sigma$ for the application of σ to the term t. A substitution σ is grounding for a term t if $t\sigma$ is ground.

Sets, sequences and multisets. We write $\mathbb{N}_n$ for the set $\{1, \ldots, n\}$. Given a set S we denote by S^* the set of finite sequences of elements from S and by $S^\#$ the set of finite multisets of elements from S. We use the superscript $^\#$ to annotate usual multiset operations, e.g. $S_1 \cup^\# S_2$ denotes the multiset union of multisets S_1, S_2. Given a multiset S we denote by $set(S)$ the set of elements in S. The sequence consisting of elements $e_1, \ldots, e_n$ will be denoted by $[e_1, \ldots, e_n]$ and the empty sequence is denoted by $[]$. We denote by $|S|$ the length, i.e., the number of elements of the sequence. We use $\cdot$ for the operation of adding an element either to the start or to the end, e.g., $e_1 \cdot [e_2, e_3] = [e_1, e_2, e_3] = [e_1, e_2] \cdot e_3$. Given a sequence S, we denote by $idx(S)$ the set of positions in S, i.e., $\mathbb{N}_n$ when S has n elements, and for $i \in idx(S)$ S_i denotes the ith element of the sequence. Set membership modulo E is denoted by $\in_E$ and defined as $e \in_E S$ iff $\exists e' \in S.\ e' =_E e$. $\subset_E$, $\cup_E$, and $=_E$ are defined for sets in a similar way. Application of substitutions are lifted to sets, sequences and multisets as expected. By abuse of notation we sometimes interpret sequences as sets or multisets; the applied operators should make the implicit cast clear.

Functions. We suppose that functions between terms are interpreted modulo E, i.e., if $x =_E y$ then $f(x) = f(y)$. Given function f we let $f(x) = \bot$ when $x \notin_E \mathbf{D}(x)$. When $f(x) = \bot$ we say that f is undefined for all $y =_E x$. We define the function $f := g[a \mapsto b]$ with $\mathbf{D}(f) = \mathbf{D}(g) \cup_E \{a\}$ as $f(x) := b$ for $x =_E a$ and $f(x) := g(x)$ for $x \neq_E a$.

3. Cryptographic calculus with local progress

We extend the Stateful Applied Pi calculus (SAPiC) [19] adding three necessary ingredients to show fairness in fair exchange protocols:

Local progress: each process needs to be reduced as far as possible. That is, until it is either waiting to receive a message, or until it reaches a replication (as we cannot replicate the process indefinitely).

Resilient channels: There is a resilient channel which guarantees message delivery, i.e., each trace is induced by at least one execution in which all messages sent were delivered.

External non-determinism: Any process $P+Q$ reduces to P' or Q' if either P reduces to P', or Q to Q'. Hence, if either P or Q are able to progress, then $P+Q$ *must* progress.

3.1. Syntax and informal semantics

SAPiC is a variant of the applied pi calculus [1]. In addition to the usual operators for concurrency, replication, communication, and name creation, SAPiC was designed for the analysis of state-based protocols and cryptographic APIs, hence it offers several constructs for reading and updating an explicit global state. For the analysis of fair-exchange protocols, we add constructs for non-deterministic choice and communication on a reliable channel to SAPiC. The resulting grammar for processes is described in Figure 1.

$$
\begin{array}{lll}
\langle P,Q \rangle ::= & 0 & \\
\mid & P \mid Q & \\
\mid & P + Q & \\
\mid & !\,P & \\
\mid & \nu n\colon \mathit{fresh};\ P & \\
\mid & \text{out}(c, N);\ P & (c \in \{\text{‘r’}, \text{‘c’}\}\colon pub) \\
\mid & \text{in}(c, N);\ P & (c \in \{\text{‘r’}, \text{‘c’}\}\colon pub) \\
\mid & \text{if } \mathit{Pred} \text{ then } P \text{ [else } Q] & \\
\mid & \text{event } F\ ;\ P & (F \in \mathcal{F}) \\
\mid & \text{insert } M,N;\ P & \\
\mid & \text{delete } M;\ P & \\
\mid & \text{lookup } M \text{ as } x \text{ in } P \text{ [else } Q] & \\
\mid & \text{lock } M;\ P & \\
\mid & \text{unlock } M;\ P &
\end{array}
$$

Figure 1: Syntax, where $M, N \in \mathcal{T}$ and $\mathit{Pred} \in \mathcal{P}$

0 denotes the terminal process. $P \mid Q$ is the parallel execution of processes P and Q and $!P$ the replication of P allowing an unbounded number of sessions in protocol executions. $P+Q$ denotes *external* non-deterministic choice, as discussed above. The construct $\nu n; P$ binds the name $n \in FN$ in P and models the generation of a fresh, random value. The processes out(c, N); P and in(c, N); P represent the output, respectively input, of message N on channel $c \in \{\text{‘c’}, \text{‘r’}\}$. There are exactly two channels, one for reliable communication, e.g., between a protocol participant and the trusted third party, and one public channel. Messages on both channels may be intercepted and altered by the adversary, however, the reliable channel guarantees that eventually, the message that was sent arrives. Readers familiar with the applied pi calculus [1] may note that we opted for the possibility of pattern matching in the input construct, rather than merely binding the input to a variable x. The process if Pred then P else Q will execute P or Q, depending on whether Pred holds. For example, if $\mathit{Pred} = equal(M, N)$, and $\phi_{equal} = x_1 \approx x_2$, then if $equal(M, N)$ then P else Q will execute P if $M =_E N$

and Q otherwise. (In the following, we will use $M = N$ as a short-hand for $equal(M, N)$.) The event construct is merely used for annotating processes and will be useful for stating security properties. For readability, we sometimes omit trailing 0 processes, respectively, else branches that consist of a 0 process.

Note that several semantics would be possible for the non-deterministic choice operator. One possibility would be a purely internal non-deterministic choice, whose semantics would correspond to the following reduction rules: $P_1 + P_2 \to P_i$ $(1 \leq i \leq 2)$. The other possibility, which we choose here, is an external choice whose semantics is defined by the rules

$$\frac{P_i \to Q}{P_1 + P_2 \to Q} \quad (1 \leq i \leq 2)$$

In this version $P_1 + P_2$ may only behave as P_1 or P_2 if the chosen process can indeed reduce, i.e., execute an action. While the external choice does complicate the translation towards Tamarin, this flavour of choice is required for modelling fair exchange protocols as we will illustrate in Example 2 below.

The remaining constructs are used to manipulate state and were introduced with SAPiC [18]. The construct insert M,N binds the value N to a key M. Successive inserts overwrite this binding, the delete M operation "undefines" the binding. The lookup M as x in P else Q allows for retrieving the value associated to M binding it to the variable x in P. If the mapping is undefined for M, the process behaves as Q. The lock and unlock constructs are used to gain or waive exclusive access to a resource M, in the style of Dijkstra's binary semaphores: if a term M has been locked, any subsequent attempt to lock M will be blocked until M has been unlocked. This is essential for writing protocols where parallel processes may read and update a common memory.

Example 2. *The following example models the responder in the ASW contract-signing protocol (cf. Section 8.2), simplified to use pattern m_1 and m_3 to match the first and the third message of the optimistic protocol.*

$$\mathrm{in}(\text{'}c\text{'}, m_1); \mathrm{out}(\text{'}c\text{'}, m_2); \begin{pmatrix} \mathrm{in}(\text{'}c\text{'}, m_3); \mathrm{out}(\text{'}c\text{'}, m_4) \\ +(\mathrm{out}(\text{'}r\text{'}, \langle m_1, m_2 \rangle); \ldots) \end{pmatrix}$$

The responder emits m_2, but is not sure to receive the response m_3, as the originator might be dishonest or the adversary might have intercepted this message. If m_3 arrives, then the process $\mathrm{in}(\text{'}c\text{'}, m_3); \mathrm{out}(\text{'}c\text{'}, m_4)$ is able to transition to $\mathrm{out}(\text{'}c\text{'}, m_4)$. If m_3 does not arrive, the message $\langle m_1, m_2 \rangle$ is transmitted on the reliable channel, to contact the TTP. This example highlights the need for external choice, as opposed to internal choice: with internal choice, the responder could simply move to the first branch $\mathrm{in}(\text{'}c\text{'}, m_3); \mathrm{out}(\text{'}c\text{'}, m_4)$ and would then be unable to contact the TTP. This would result in an unsound model. Using external choice, however, moving to the first branch is only possible if m_3 is indeed available for input.

3.2. Semantics

Frames and deduction. Before giving the formal semantics of SAPiC, we introduce the notions of frame and deduction. A *frame* consists of a set of fresh names $\tilde{n}$ and a substitution σ, and is written $\nu\tilde{n}.\sigma$. Intuitively, a frame represents the sequence of messages that have been observed by an adversary during a protocol execution and secrets $\tilde{n}$ generated by the protocol, a priori unknown to the adversary. Deduction models the capacity of the adversary to compute new messages from the observed ones.

Definition 1 (Deduction)**.** *We define the deduction relation $\nu\tilde{n}.\sigma \vdash t$ as the smallest relation between frames and terms defined by the deduction rules in Figure 2.*

Operational semantics. We can now define the operational semantics of our calculus. The semantics is defined by a labelled transition relation between process configurations. A *process configuration* is a 6-tuple $(\mathcal{E}, \mathcal{S}, \mathcal{P}, \sigma, \mathcal{L}, \mathcal{U})$ where

- $\mathcal{E} \subseteq FN$ is the set of fresh names generated by the processes;
- $\mathcal{S} : \mathcal{M}_\Sigma \to \mathcal{M}_\Sigma$ is a partial function modeling the store;
- $\mathcal{P}$ is a multiset of ground processes representing the processes executed in parallel;
- σ is a ground substitution modeling the messages output to the environment;
- $\mathcal{L} \subseteq \mathcal{M}_\Sigma$ is the set of currently active locks
- $\mathcal{U} \subseteq \mathcal{M}_\Sigma^\#$ is the multiset of messages pending delivery on the public (resilient) channel

The transition relation is defined by the rules in Figure 3. Transitions are labelled by sets of ground facts. For readability, we omit empty sets and brackets around singletons, i.e., we write $\to$ for $\xrightarrow{\emptyset}$ and $\xrightarrow{f}$ for $\xrightarrow{\{f\}}$. We write $\to^*$ for the reflexive, transitive closure of $\to$ (the transitions that are labelled by the empty sets) and write $\overset{f}{\Rightarrow}$ for $\to^* \xrightarrow{f} \to^*$. We can now define the set of traces, i.e., possible executions that a process admits. As we are interested in liveness properties we will only consider the set of *progressing* traces, that is traces that end with a *final* state. Intuitively, a state is final if all messages on resilient channels have been delivered and the process is *blocking*.

Definition 2. *Given a ground process P we define the predicate* blck *as follows*

$$\mathsf{blck}(P) \hat{=} \begin{cases} \top, & \text{if } P = 0,\ P = !Q \text{ or } P = \mathrm{in}(c, m); Q \\ \mathsf{blck}(P_1) \wedge \mathsf{blck}(P_2), & \text{if } P = P_1 + P_2 \\ \bot, & \text{otherwise} \end{cases}$$

Intuitively a process will always execute completely, except if it is a replication or when it blocks for external reasons, i.e., it is awaiting input on (the public or the resilient) channel.

Definition 3 (Traces of P)**.** *Given a ground process P we define the* set of progressing traces of *P as*

$$\frac{a \in FN \cup PN \quad a \notin \tilde{n}}{\nu\tilde{n}.\sigma \vdash a} \text{DNAME} \qquad \frac{\nu\tilde{n}.\sigma \vdash t \quad t =_E t'}{\nu\tilde{n}.\sigma \vdash t'} \text{DEQ}$$

$$\frac{x \in \mathbf{D}(\sigma)}{\nu\tilde{n}.\sigma \vdash x\sigma} \text{DFRAME} \qquad \frac{\nu\tilde{n}.\sigma \vdash t_1 \cdots \nu\tilde{n}.\sigma \vdash t_n \quad f \in \Sigma^k \setminus \Sigma^k_{priv}}{\nu\tilde{n}.\sigma \vdash f(t_1, \ldots, t_n)} \text{DAPPL}$$

Figure 2: Deduction rules.

$$traces^{ppi}(P) = \Big\{(F_1, \ldots, F_n) \mid c_0 \stackrel{F_1}{\Longrightarrow}_* \ldots \stackrel{F_n}{\Longrightarrow}_* c_n \wedge final(c_n)\Big\}, \text{ where}$$

$c_0 = (\emptyset, \emptyset, \{ P \}, \emptyset, \emptyset, \emptyset)$, *and* $final(\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$ *iff* $\mathcal{U}_n = \emptyset$ *and* $\mathsf{blck}(P)$ *for all* $P \in \mathcal{P}$.

3.3. Discussion

Our calculus supports two channels: 'c' is public and not resilient, while 'r' is public and resilient. It is easy to model an arbitrary number of resilient channels by consistently using pattern matching, e.g., one channel per session id sid and party A is encoded by using $\text{out}(\text{'r'}, \langle A, sid, m\rangle)$ and $\text{in}(\text{'r'}, \langle A, sid, m\rangle)$ throughout the process modelling A in session sid.

Unlike in the original SAPiC [19], we do not support private channels for the following reason. Suppose a process is reduced to $P = \text{out}(s, m); P'$, and s is a supposedly secret channel name. As there is no matching input $\text{in}(s, m)$, internal communication is not an option, and thus, the process can only reduce further if the adversary was able to deduce s. Hence, whether P is final would depend on whether s is private, i.e., deducible by the adversary, which would significantly complicate the translation.

4. Labelled multiset rewriting

We now recall the syntax and semantics of labelled multiset rewriting rules, which constitute the input language of the Tamarin tool [26].

Definition 4 (Multiset rewrite rule)**.** *A labelled multiset rewrite rule* ri *is a triple* (l, a, r), $l, a, r \in \mathcal{F}^*$, *written* $l \;-[\; a \;]\!\!\rightarrow\; r$. *We call* $l = prems(ri)$ *the premises,* $a = actions(ri)$ *the actions, and* $r = conclusions(ri)$ *the conclusions of the rule.*

A labelled multiset rewriting system is a set of labelled multiset rewrite rules R, which satisfy some side-conditions which we leave out for brevity. There is one distinguished rule FRESH which is the only rule allowed to have Fr-facts on the right-hand side

$$\text{FRESH} : [] \;-[]\!\!\rightarrow [\mathsf{Fr}(x : fresh)].$$

The semantics of the rules is defined by a labelled transition relation.

Definition 5 (Labelled transition relation)**.** *Given a multiset rewriting system* R *we define the* labeled transition relation $\rightarrow_R \subseteq \mathcal{G}^\# \times \mathcal{P}(\mathcal{G}) \times \mathcal{G}^\#$ *as*

$$S \stackrel{a}{\longrightarrow}_R ((S \setminus^\# lfacts(l)) \cup^\# r)$$

if and only if $l \;-[\; a \;]\!\!\rightarrow\; r \in_E ginsts(R \cup \text{FRESH})$, $lfacts(l) \subseteq^\# S$ *and* $pfacts(l) \subseteq S$.

Definition 6 (Executions)**.** *Given a multiset rewriting system* R *we define its set of executions as*

$$exec^{msr}(R) = \Big\{\emptyset \stackrel{A_1}{\longrightarrow}_R \ldots \stackrel{A_n}{\longrightarrow}_R S_n \mid \forall i, j \in \mathbb{N}_n, a. \; (S_{i+1} \setminus^\# S_i) = \{ \mathsf{Fr}(a) \}^\# \wedge (S_{j+1} \setminus^\# S_j) = \{ \mathsf{Fr}(a) \}^\# \Rightarrow i = j\Big\}$$

The set of executions consists of transition sequences that respect freshness, i. e., for a given name a the fact $\mathsf{Fr}(a)$ is only added once, or in other words the rule FRESH is at most fired once for each name. We define the set of traces in a similar way as for processes.

Definition 7 (Traces)**.** *The set of traces is defined as*

$$traces^{msr}(R) = \Big\{(A_1, \ldots, A_n) \mid \; \forall\, 0 \leq i \leq n. \; \emptyset \stackrel{A_1}{\Longrightarrow}_R \ldots \stackrel{A_n}{\Longrightarrow}_R S_n \in exec^{msr}(R)\Big\}$$

where $\stackrel{A}{\Longrightarrow}_R$ *is defined as* $\stackrel{\emptyset}{\longrightarrow}^*_R \stackrel{A}{\longrightarrow}_R \stackrel{\emptyset}{\longrightarrow}^*_R$ *for* $A \neq \emptyset$.

Note that both for processes and multiset rewrite rules the set of traces is a sequence of sets of facts.

5. Security Properties

In the Tamarin tool [26], security properties are described in an expressive two-sorted first-order logic. The sort $temp$ is used for time points, $\mathcal{V}_{temp}$ are the temporal variables.

Definition 8 (Trace formulas)**.** *A trace atom is either false* $\perp$, *a term equality* $t_1 \approx t_2$, *a timepoint ordering* $i \lessdot j$, *a timepoint equality* $i \doteq j$, *or an action* $F@i$ *for a fact* $F \in \mathcal{F}$ *and a timepoint* i. *A trace formula is a first-order formula over trace atoms.*

As we will see in our case studies, this logic is expressive enough to analyze a variety of security properties, including liveness properties.

To define the semantics, let each sort s have a domain $\mathbf{D}(s)$. $\mathbf{D}(temp) = \mathcal{Q}$, $\mathbf{D}(msg) = \mathcal{M}$, $\mathbf{D}(fresh) = FN$, and

Standard operations:

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{0\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P|Q\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P,Q\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P+Q\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{A} (\mathcal{E}',\mathcal{S}',\mathcal{P}',\sigma',\mathcal{L}',\mathcal{U}')$$
$$\text{if } (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L},\mathcal{U})\xrightarrow{A}(\mathcal{E}',\mathcal{S}',\mathcal{P}',\sigma',\mathcal{L}',\mathcal{U}') \text{ or } (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{Q\},\sigma,\mathcal{L},\mathcal{U})\xrightarrow{A}(\mathcal{E}',\mathcal{S}',\mathcal{P}',\sigma',\mathcal{L}',\mathcal{U}')$$
$$\text{where } \mathcal{P}'=\mathcal{P} \text{ or } \mathcal{P}'=\mathcal{P}\cup^{\#}\{\,P'\,\}^{\#} \text{ for some } P'\neq P$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{!P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{!P,P\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\nu a;P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E}\cup\{a'\},\mathcal{S},\mathcal{P}\cup^{\#}\{P\{^{a'}/_{a}\}\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } a' \text{ is fresh}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(M)} (\mathcal{E},\mathcal{S},\mathcal{P},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \nu\mathcal{E}.\sigma\vdash M$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{out}(\text{‘r’},N);P\},\sigma,\mathcal{L},\mathcal{U}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma\cup\{^{N}/_{x}\},\mathcal{L},\mathcal{U}\cup^{\#}\{\,N\,\}^{\#}) \quad \text{if } x \text{ is fresh}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{in}(\text{‘r’},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(N\tau)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\tau\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \exists\tau.\ \tau \text{ is grounding for } N, \nu\mathcal{E}.\sigma\vdash N\tau$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{in}(\text{‘r’},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(N\tau)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\tau\},\sigma,\mathcal{L},\mathcal{U}\setminus^{\#}\{\,N'\,\}^{\#}) \quad \text{if } \exists N',\tau.\ N'\in\mathcal{U}, \tau \text{ is grounding for } N, N\tau=_E N'$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{out}(\text{‘c’},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(\text{‘c’})} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma\cup\{^{N}/_{x}\},\mathcal{L},\mathcal{U}) \quad \text{if } x \text{ is fresh}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{in}(\text{‘c’},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(\langle\text{‘c’},N\tau\rangle)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\tau\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \exists\tau.\ \tau \text{ is grounding for } N, \nu\mathcal{E}.\sigma\vdash N\tau$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup\{\text{if } pr(M_1,\ldots,M_n) \text{ then } P \text{ else } Q\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{P\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \phi_{pr}\{^{M_1}/_{x_1},\ldots,^{M_n}/_{x_n}\} \text{ is satisfied}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup\{\text{if } pr(M_1,\ldots,M_n) \text{ then } P \text{ else } Q\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{Q\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \phi_{pr}\{^{M_1}/_{x_1},\ldots,^{M_n}/_{x_n}\} \text{ is not satisfied}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup\{\text{event}(F);\,P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{F} (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{P\},\sigma,\mathcal{L},\mathcal{U})$$

Operations on global state:

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{insert } M,N;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S}[M\mapsto N],\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{delete } M;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S}[M\mapsto\bot],\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q\,\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\{V/x\}\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \mathcal{S}(M)=_E V \text{ is defined}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q\,\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{Q\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \mathcal{S}(M) \text{ is undefined}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lock } M;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L}\cup\{\,M\,\},\mathcal{U}) \quad \text{if } M\not\in_E\mathcal{L}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{unlock } M;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L}\setminus\{\,M' \mid M'=_E M\,\},\mathcal{U})$$

Figure 3: Operational semantics

$\mathbf{D}(pub)=PN$. A function $\theta:\mathcal{V}\to\mathcal{M}\cup\mathcal{Q}$ is a valuation if it respects sorts, i.e., $\theta(\mathcal{V}_s)\subset\mathbf{D}(s)$ for all sorts s. If t is a term, $t\theta$ is the application of the homomorphic extension of θ to t.

Definition 9 (Satisfaction relation)**.** *The satisfaction relation $(tr,\theta)\vDash\varphi$ between a trace tr, a valuation θ, and a trace formula φ is defined as follows:*

$(tr,\theta)\vDash\bot$		*never*
$(tr,\theta)\vDash F@i$	$\iff$	$\theta(i)\in idx(tr)\wedge F\theta\in_E tr_{\theta(i)}$
$(tr,\theta)\vDash i\lessdot j$	$\iff$	$\theta(i)<\theta(j)$
$(tr,\theta)\vDash i\doteq j$	$\iff$	$\theta(i)=\theta(j)$
$(tr,\theta)\vDash t_1\approx t_2$	$\iff$	$t_1\theta=_E t_2\theta$
$(tr,\theta)\vDash\neg\varphi$	$\iff$	*not* $(tr,\theta)\vDash\varphi$
$(tr,\theta)\vDash\varphi_1\wedge\varphi_2$	$\iff$	$(tr,\theta)\vDash\varphi_1$ *and* $(tr,\theta)\vDash\varphi_2$
$(tr,\theta)\vDash\exists x:s.\varphi$	$\iff$	*there is* $u\in\mathbf{D}(s)$ *such that* $(tr,\theta[x\mapsto u])\vDash\varphi$.

For readability, we define $t_1\gtrdot t_2$ as $\neg(t_1\lessdot t_2\vee t_1\doteq t_2)$ and $(\lessdot\!\!\!\!=,\not\doteq,\gtrdot\!\!\!\!=)$ as expected. We also use classical notational shortcuts such as $t_1\lessdot t_2\lessdot t_3$ for $t_1\lessdot t_2\wedge t_2\lessdot t_3$ and $\forall i\leq j.\ \varphi$ for $\forall i.\ i\leq j\to\varphi$. When φ is a ground formula

we sometimes simply write $tr \vDash \varphi$ as the satisfaction of φ is independent of the valuation.

Definition 10 (Validity, satisfiability)**.** *Let $Tr \subseteq (\mathcal{P}(\mathcal{G}))^*$ be a set of traces. A trace formula φ is said to be* valid *for Tr (written $Tr \vDash^\forall \varphi$) if for any trace $tr \in Tr$ and any valuation θ we have that $(tr, \theta) \vDash \varphi$.*

A trace formula φ is said to be satisfiable *for Tr, written $Tr \vDash^\exists \varphi$, if there exist a trace $tr \in Tr$ and a valuation θ such that $(tr, \theta) \vDash \varphi$.*

Note that $Tr \vDash^\forall \varphi$ iff $Tr \nvDash^\exists \neg\varphi$. Given a multiset rewriting system R we say that φ is valid, written $R \vDash^\forall \varphi$, if $traces^{msr}(R) \vDash^\forall \varphi$. We say that φ is satisfied in R, written $R \vDash^\exists \varphi$, if $traces^{msr}(R) \vDash^\exists \varphi$. Similarly, given a ground process P we say that φ is valid, written $P \vDash^\forall \varphi$, if $traces^{ppi}(P) \vDash^\forall \varphi$, and that φ is satisfied in P, written $P \vDash^\exists \varphi$, if $traces^{ppi}(P) \vDash^\exists \varphi$.

Example 3. *In Section 8, the following trace formula is used to express timeliness for the originator, i.e., whenever originator 'a' runs the protocol with 'b' on a contract 't' in session 'sid', unless 'a' will be corrupted at some point, she will eventually reach either a contract in this session or receive notification that it has been aborted.*

$$\begin{aligned}\forall i : temp, a, b, t, sid : msg.\\ Start_A(a,b,t,sid)@i \Rightarrow &(\exists j.Contract_A(a,b,t,sid)@j)\\ &|(\exists j.Abort_A(a,b,t,sid)@j)\\ &|(\exists j.Corrupt(a)@j)\text{''}\end{aligned}$$

6. A translation from processes into multiset rewrite rules

In this section, we define a translation from a process P into a set of multiset rewrite rules $[\![P]\!]$ and a translation on trace formulas such that $P \models^\forall \varphi$ if and only if $[\![P]\!] \models^\forall [\![\varphi]\!]$. Note that the result also holds for satisfiability as an immediate consequence. For a rather expressive subset of trace formulas (see [26] for the exact definition of the fragment), checking whether $[\![P]\!] \models^\forall [\![\varphi]\!]$ can then be discharged to the Tamarin prover that we use as a backend. Except for local progress, resilient channels and NDC, the other elements of the translation have been discussed in previous work [19].

6.1. Progress function

In Section 3, we have defined local progress axiomatically in terms of the final state to be reached. The progress function that we use for our translation gives a more constructive understanding. In this section, we will give an intuition of how this function works and illustrate the subtle interplay between non-deterministic choice and local progress. We postpone the formal definition to Appendix A.

When a process is in a certain position p, the progress function π defines the maximal follow-up positions that the process can reach on its own. All traces that do not reach maximal positions will later be ruled out by the means of an axiom. As we will see below, the progress function maps a position to a set of sets of positions. In the simplest case a position in a process can have a unique position it must progress to.

Example 4. *Let*

$$P = \text{event } A; \text{in}(\text{'}c\text{'}, m); \text{event } B; 0.$$

Initially, the process P must reduce to

$$P|_1 = \text{in}(\text{'}c\text{'}, m); \text{event } B; 0$$

i.e., raise A. However, the process will be blocked as it needs to wait for an input. Once P can be reduced to $P|_{111} =$ event $B; 0$, e.g., because the adversary sends a message, it must continue to reduce to 0. We would therefore define the progress function for P such that $\pi([]) = \{\{1\}\}$ and $\pi(11) = \{\{111\}\}$

In general, a process needs to progress until it reaches a *blocking* process, as defined in Definition 2. We call a position blocking in P, if $P|_p$ is blocking. If P is clear from context, we only call the position blocking. Note that in case of a non-deterministic choice, the process $P = P_1 + P_2$ is only blocking if both P_1 and P_2 are blocking too. If one of the P_i is non-blocking, the process will progress in that branch. Therefore, progress can never reach the position directly below a $+$. In particular, if NDC operators are nested (which is useful, e.g., to express an n-fold choice), several positions need to be 'jumped over', figuratively speaking.

Moreover, in case of a parallel composition, the progress function expects the process to progress to the next blocking position in each of the parallel branches.

Example 5. *Consider the process*

$$P = (\text{event } A; 0) \mid (\text{event } B; 0)$$

This process is expected to raise both events A and B. Therefore, $\pi([]) = \{\{11\}, \{21\}\}$.

A process $P = P_1 + P_2$ allows to either execute P_1 or P_2 but not both. Unlike parallel composition where P must progress in both branches, we ensure that P processes in either P_1 or P_2.

Example 6. *Consider the process*

$$P = (\text{event } A; 0) + (\text{event } B; 0)$$

To express that P must either raise event A or event B, we define $\pi([]) = \{\{11, 21\}\}$.

We are now ready to explain why the progress function requires to return a set of sets of positions. When

$$\pi(p) = \{A_1, \ldots A_n\},$$

process P must transition p to some position in A_i for each $1 \le i \le n$. Intuitively, each A_i corresponds to a parallel branch. The positions in each A_i are the mutually exclusive positions due to the $+$ operator. We hence require that a

process at position p executes until it reaches one blocking position for each A_i.

Example 7. *Consider the process*

$$(\text{event } A; 0 \mid \text{event } B; 0) + (\text{event } C; 0 \mid \text{event} D; 0)$$

and denote the leaf position below event A *be called* p_A *and similarly for other events. We have that* $\pi([]) = \{\{p_A, p_C\}, \{p_A, p_D\}, \{p_B, p_C\}, \{p_B, p_D\}\}$

Example 8. *The following example illustrates the difficulty in defining* π *due to non-deterministic choice.*

$$\begin{aligned} P = \quad & (\text{in}(`c', m); \text{event } A; 0) + \\ & (\text{out}(`r', r_1); \text{event } B; \text{in}(`r', r_2); \text{event } C; 0) \end{aligned}$$

While the left branch starts with a blocking position, progress is possible in the right branch. As π *aims to capture the guarantee that a process will progress until no action is directly available, we have to consider two cases: a) If the input in the first branch is not available,* P *has no choice but to progress to the second branch, which is non-blocking. b) If the input in the first branch is available, then* P *has to reduce to the next blocking position in the first branch. Thus* $\pi([]) = \{\{111, 211\}\}$*, and* $\pi(2111) = \{\{21111\}\}$*. This situation appears in all contract-signing protocols we verify: either a message appears on the public channel and we can proceed to* A*, or the process eventually gives up and contacts the TTP on the reliable channel. If the TTP is implemented well, then due to local progress and the fact that messages sent on the reliable channel are eventually delivered, the position below event* B *(*2111*) can be reached, which triggers progress up to event* C*.*

When π is the progress function for P we will denote by $\mathsf{From}(P)$ the domain of π and $\mathsf{To}(P)$ the set of positions that appear in the image of π. $\mathsf{From}(P)$ is the set of positions where progress starts, roughly, the non-blocking positions that directly follow a blocking position and possibly $[]$ (if $[]$ is non blocking). We also show that for any position $q \in \mathsf{To}(P)$ there is a unique position $p \in \mathsf{From}(P)$ such that q appears in $\pi(p)$ and denote the function that maps q to p by π^{-1}. The formal definitions of these sets and the progress function are rather technical and can be found in the long version.

6.2. Definition of the translation of processes

The translation is defined on *well-formed processes*, i.e., ground processes that do not contain reserved variables or reserved facts, in which every variable is under the scope of exactly one binder and which fulfils a syntactic criterion on locks. We leave the details for the full version, as none of these conditions constitutes a limitation for this work (as has thoroughly been discussed in [19]).

Definition 11. *Given a well-formed ground process* P *we define the labelled multiset rewriting system* $[\![P]\!]$ *as*

$$\text{MD} \cup \{\textsc{Init}, \text{MID}\} \cup [\![\overline{P}, [], []]\!]$$

where

- *the rule* $\textsc{Init}$ *is defined as*

$$\textsc{Init} : [\mathsf{Fr}_{[]}] \mathrel{-\![} \text{Init}, \mathit{ProgFrom}_{[]} \mathrel{]\!\!\rightarrow} [\mathsf{state}_{[]}()],$$

- *the rule* MID *is defined as*

$$\text{MID} : [\mathsf{Fr}(x)] \mathrel{-\![} \quad \mathrel{]\!\!\rightarrow} [\mathsf{MID}_{\mathsf{rcv}}(x), \mathsf{MID}_{\mathsf{snd}}(x)]$$

- $[\![P, p, \tilde{x}]\!]$ *is defined inductively for process* P*, position* $p \in \mathbb{N}^*$ *and sequence of variables* $\tilde{x}$ *in Figure 4.*
  For brevity, we use the following syntactic shortcuts:

$$\mathit{ProgFrom}_p \hat{=} \begin{cases} \mathit{ProgFrom}_p(\mathit{prog}_p) & \text{if } p \in \mathsf{From}(P) \\ [] & \text{otherwise} \end{cases}$$

$$\mathit{ProgTo}_p \hat{=} \begin{cases} \mathit{ProgTo}_p(\mathit{prog}_{\pi^{-1}(p)}) & \text{if } p \in \mathsf{To}(P) \\ [] & \text{otherwise} \end{cases}$$

$$\mathsf{Fr}_p \hat{=} \begin{cases} \mathsf{Fr}(\mathit{prog}_p) & \text{if } p \in \mathsf{From}(P) \\ [] & \text{otherwise} \end{cases}$$

$$\tilde{x}_p \hat{=} \begin{cases} \tilde{x} \cup^{\#} \{\mathit{prog}_p\}^{\#} & \text{if } p \in \mathsf{From}(P) \\ \tilde{x} & \text{otherwise} \end{cases}$$

The core of the translation builds on [19]. The message deduction rules MD consist of four rules for message output, message input, application of (non-private) function symbols, and creation of fresh values. In the definition of $[\![P, p, \tilde{x}]\!]$, we intuitively use the family of facts state_p to indicate that the process is currently at position p in its syntax tree. A fact state_p will indeed be true in an execution of these rules whenever some instance of P_p (i.e. the process defined by the subtree at position p of the syntax tree of P) is in the multiset $\mathcal{P}$ of the process configuration.

We will now comment on the main changes w.r.t. [19]. The translation of a NDC does not produce a rule, but rewrites the positions of the required state-fact in the first rules of both its child processes, with the effect that the NDC step is skipped to proceed to either one. Input and output on the resilient channel require the fact $\mathsf{MID}_{\mathsf{snd}}$, respectively $\mathsf{MID}_{\mathsf{rcv}}$, both of which can be instantiated with the rule MID. Each instantiation can be used but once and assures that each message sent has a unique identifier, even if a message with the same content has been sent before. Thus, the axiom α_{resil} can enforce that a message sent must be received, for each instance of a message. Finally, we annotate the rules with $\mathit{ProgFrom}$ and ProgTo facts. The α_{prog} axiom will use these actions to enforce progress.

6.3. Definition of the translation of trace formulas

A trace formula φ is well-formed if no reserved variable nor a reserved fact appear in φ.

Definition 12. *Given a well-formed trace formula* φ *we define*

$$[\![\varphi]\!]_\forall := \alpha \Rightarrow \varphi \qquad \text{and} \qquad [\![\varphi]\!]_\exists := \alpha \wedge \varphi$$

where α *is defined in Figure 5.*

$$
\begin{array}{rcl}
[\![0,p,\tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] -\![\ \ ]\!\!\rightarrow []\} \\
[\![P \mid Q,p,\tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] -\![\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathit{ProgTo}_{p\cdot 2}\ ]\!\!\rightarrow [\mathsf{state}_{p\cdot 1}(\tilde{x}_p), \mathsf{state}_{p\cdot 2}(\tilde{x}_p)]\} \\
 & & \cup [\![P,p\cdot 1,\tilde{x}_p]\!] \cup [\![Q,p\cdot 2,\tilde{x}_p]\!] \\
[\![P+Q,p,\tilde{x}]\!] & = & [\![P,p\cdot 1,\tilde{x}]\!] \left\{{}^{\mathsf{state}_p(\tilde{x})}/{}_{\mathsf{state}_{p\cdot 1}(\tilde{x})}\right\} \cup [\![Q,p\cdot 2,\tilde{x}]\!] \left\{{}^{\mathsf{state}_p(\tilde{x})}/{}_{\mathsf{state}_{p\cdot 2}(\tilde{x})}\right\} \\
[\![!P,p,\tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] -\![\ \mathit{ProgTo}_{p\cdot 1}\ ]\!\!\rightarrow [!\mathsf{state}_p^{\mathsf{semi}}(\tilde{x})], \\
 & & [!\mathsf{state}_p^{\mathsf{semi}}(\tilde{x})] -\![\ \ ]\!\!\rightarrow [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P,p\cdot 1,\tilde{x}]\!] \\
[\![\mathrm{in}(\text{‘r’},m);P,p,\tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(m), \mathsf{Fr}_p, \mathsf{MID}_{\mathsf{rcv}}(\mathit{mid})] -\![\ \mathit{ProgTo}_{p\cdot 1}, \mathrm{Receive}(\mathit{mid},m), \mathrm{InEvent}(m)\ ]\!\!\rightarrow \\
 & & \quad [\mathsf{state}_{p\cdot 1}(\tilde{x}_p \cup \mathit{vars}(m))]\} \\
 & & \cup [\![P,p\cdot 1,\tilde{x}\cup \mathit{vars}(m)]\!] \\
[\![\mathrm{out}(\text{‘c’},N);P,p,\tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(\text{‘c’})] -\![\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{InEvent}(\text{‘c’})\ ]\!\!\rightarrow [\mathsf{state}_{p\cdot 1}(\tilde{x}), \mathsf{Out}(N)]\} \\
 & & \cup [\![P,p\cdot 1,\tilde{x}_p]\!]
\end{array}
$$

Figure 4: Translation of processes: definition of $[\![P,p,\tilde{x}]\!]$. (The rules for νn, input/output on the public channel, conditional branching, events, locks, unlocks and state manipulation can be found in the full version.)

$\alpha := \alpha_{init} \wedge \alpha_{pred} \wedge \alpha_{noteq} \wedge \alpha_{in} \wedge \alpha_{notin} \wedge \alpha_{lock} \wedge \alpha_{inev} \wedge \alpha_{resil} \wedge \alpha_{prog}$ and

$$
\begin{array}{rcrl}
\alpha_{init} & := & \forall i,j. & \mathrm{Init}()@i \wedge \mathrm{Init}()@j \Longrightarrow i \doteq j \\
\alpha_{resil} & := & \forall x,y,t_1. & \mathrm{Send}(x,y)@t_1 \Longrightarrow \exists t_2.\ \mathrm{Receive}(x,y)@t_2 \wedge t_1 \leq t_2 \\
\alpha_{prog} & := & \bigwedge_{a\in \mathsf{From}(P)\wedge B\in\pi(a)} & \left\{\forall l,t_1.\ \mathit{ProgFrom}_a(l)@t_1 \Longrightarrow \exists t_2.\ \bigvee_{b\in B}(\mathit{ProgTo}_b(l)@t_2)\right\}
\end{array}
$$

Figure 5: Definition of α. ($\alpha_{pred}, \alpha_{noteq}, \alpha_{in}, \alpha_{notin}, \alpha_{lock}, \alpha_{inev}$ are defined in the full version.)

The axiom α_{resil} is a straightforward formalisation of the intuition that each message sent on the resilient channel ought to arrive at some point. Progress is enforced via the axiom α_{prog}, which directly derives from the progress function. Recall that $\pi(p)$ for some position p encodes the positions to be reached in conjunctive normal form. Similiar to our use of locks, we annotate each parting position p with a fresh nonce which re-appears in each position in $\pi(p)$. Whenever there is an action $\mathit{ProgFrom}$ in the trace, the existence of a $\mathit{ProgFrom}$ step is derived from the axiom α_{prog}. This action carries the fresh value chosen at the $\mathit{ProgFrom}$ position, which Tamarin identifies correctly merging these two steps along with the intermediary positions.

7. Proof of correctness

The correctness of our translation is stated by the following theorem.

Theorem 1. *Given a well-formed ground process P and a well-formed trace formula φ we have that*

$$\mathit{traces}^{ppi}(P) \vDash^\star \varphi \text{ iff } \mathit{traces}^{msr}([\![P]\!]) \vDash^\star [\![\varphi]\!]_\star$$

where $\star$ is either $\forall$ or $\exists$.

We here give an overview of the main propositions and lemmas needed to prove Theorem 1. Detailed proofs are given in the full version [4]. We first define additional notations:

- given a process P, we write $\mathit{traces}^{pi}(P)$ (as opposed to $\mathit{traces}^{ppi}(P)$) for the set of traces of P without requiring progress (but we do require resilience);
- given a trace formula α and a set of traces Tr, $\mathit{filter}_\alpha(Tr)$ denotes the subset of Tr on which α holds;
- given a trace tr and a set of facts $\mathcal{F}$ we write $\mathit{hide}_{\mathcal{F}}(tr)$ for the trace obtained by removing any fact in $\mathcal{F}$ and lift this operation to sets of traces as expected.

We can now adopt the main lemma of the previous translation [19], which is relating the set of traces of a process P and the set of traces of its translation into multiset rewrite rules. The set $\mathcal{F}_{res}$ denotes the set of reserved facts used in the translation and that may not appear in processes (see [19] for details). Note that this adoption does not yet take into account the progress axiom. To increase modularity we chose to treat this argument separately.

Lemma 1 (Adaptation of [19])**.** *For all P well-formed with respect to this paper's Definition,*

$$\mathit{traces}^{pi}(P) = \mathit{hide}_{\mathcal{F}_{res}}(\mathit{filter}_{\alpha\setminus\alpha_{prog}}(\mathit{traces}^{msr}([\![P]\!]))).$$

Proof sketch. The proof is largely similar to the one presented in earlier work [19]. There are two main changes.

- Sending and receiving messages on the resilient channels requires correct bookkeeping, i. e., we show that at any point of the execution, the multiset of undelivered messages equals the multiset of pairs of messages and message ids (mid, see rule MID) for which a Send-action appears in the trace, but not a Receive-action.

- We handle (possibly nested) non-deterministic choice. □

In order to show the same property for $traces^{ppi}$ and α including α_{prog}, we have to show that, at any point, if and only if no process can be further reduced, is α, including α_{prog} preserved.

Lemma 2 (Correctness of α_{prog})**.** *The following two statements are equivalent for all $E_1, \ldots, E_m$:*

$$\exists s_1, \ldots, s_m.(s_0 \xRightarrow{E_1} \cdots \xRightarrow{E_m} (\mathcal{E}_m, \mathcal{S}_m, \mathcal{P}_m, \sigma_m, \mathcal{L}_m, \mathcal{U}_m)) \wedge \forall Q \in \mathcal{P}_m.\mathsf{blck}(Q)$$

iff

$$\exists S_1, \ldots, S_n, F_1, \ldots, F_n.\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in exec^{msr}(\llbracket P \rrbracket) \wedge (E_1, \ldots, E_m) = hide_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \wedge ((F_1, \ldots, F_n)) \vDash \alpha.$$

Proof sketch. The main argument relies only on the correctness of the progress function, (Lemmas 5 to 11 in the full version). For the first direction, if there is a process that is non-blocking, there is a $ProgFrom$-action not yet 'resolved' by a $ProgTo$-action, otherwise α_{prog} would hold. Conversely, where α_{prog} does not hold, one can point to a $ProgFrom$ that is not 'resolved', which identifies a position that must be a prefix of some position in the process that has not been further resolved. □

Combining Lemmas 1 and 2, we can show our main lemma.

Lemma 3 (trace-equivalence)**.** *For all well-formed P, then*

$$traces^{ppi}(P) = hide_{\mathcal{F}_{res}}(filter_{\alpha}(traces^{msr}(\llbracket P \rrbracket))).$$

Proof sketch. The idea is to define $traces^{ppi}(P)$ in terms of the set difference between $traces^{pi}(P)$ and non-final traces. As the negation of Lemma 2 shows equivalence between non-final SAPiC executions and msr executions that are filtered, the rest of the proof is a set-theoretical transformation. □

The main theorem follows Lemma 3, and Propositions 3 and 2 (cf. the full version).

Proof of Theorem 1.

$$\begin{aligned} & traces^{ppi}(P) \vDash^{\star} \varphi \\ \Leftrightarrow\ & hide_{\mathcal{F}_{res}}(filter_{\alpha}(traces^{msr}(\llbracket P \rrbracket))) \vDash^{\star} \varphi && \text{by Lemma 3} \\ \Leftrightarrow\ & filter_{\alpha}(traces^{msr}(\llbracket P \rrbracket)) \vDash^{\star} \varphi && \text{by Prop. 3} \\ \Leftrightarrow\ & traces^{msr}(\llbracket P \rrbracket) \vDash^{\star} \llbracket \varphi \rrbracket_{\star} && \text{by Prop. 2} \end{aligned}$$

□

The axiom α_{inev} within α has turned out to slow down verification time, which is why we have shown that for a particular class of formulas it is possible to remove it. We show in the full version (Theorem 2) that this is sound for all security properties we are interested in.

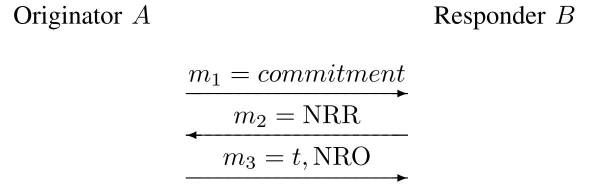
8. Case studies

In this section we present the analyses of several case studies using our extension of SAPiC/tamarin toolchain. The obtained results, obtained using 16 2.5GhZ Intel Xeon E7-8867 cores and 1.5TB available RAM are summarised in Tables 1 and 2. The implementation and SAPiC models are part of the tamarin-prover repository [2].

8.1. A first toy protocol

Our first case study is a fair non-repudiation protocol introduced in [20] to motivate the need for timeliness in addition to fairness. We will also use this protocol to discuss our modelling of fairness, timeliness and corruption.

Protocol description. The protocol consists of two sub-protocols: a main protocol and a recovery protocol. In the main protocol, 3 messages are exchanged.



Toy protocol: honest protocol run

In order to concentrate on the message flow, we do not give the details of these messages, which can be found in [20]. The first message m_1 is a commitment of A to send some text t. B replies by sending m_2 which represents a non-repudiation of receipt (NRR) evidence. Finally, message m_3 contains t and a non-repudiation of origin (NRO) proof. An obvious fairness problem arises when B does not receive m_3. In this case, he may contact the TTP with a resolve request (which includes m_1). m_1 contains enough information for the TTP to recover t and produce a non-repudiation of origin evidence on behalf of A. The TTP sends the evidence and t to B. The TTP also sends a non-repudiation of receipt evidence to A on behalf of B: this is important as a dishonest B could otherwise request a resolve after having received m_1 without sending m_2.

The processes used to model the roles of A, B and the TTP are given in Figure 7. The definitions of the messages m_i and r_i are available in our example files. The processes P_A and P_B use the NDC operator to model the possible branching in case of a recovery. Note that, unlike the more complex ASW and GJM examples, here the model of the TTP neither requires NDC nor persistent state (to store the status of the protocol). The processes are annotated with events that allow us to define security properties. Even though the toy protocol was designed to exchange non-repudiation evidences we will refer to the items to be exchanged as contracts (these evidences may be seen as a kind of contract).

2. https://github.com/tamarin-prover/tamarin-prover

TABLE 1: Case studies and results: ✓ denotes successful verification, while ✗ denotes we discovered an attack. ∞ means that the verification procedure diverges.

	ASW		ASW (mod.)		GJM		GJM (mod.)		toy example	
property	type	time	type	time	type	time	type	time	type	time
timeliness (A)	✓	1:40min	✓	1:38min	✓	0:46min	✓	6:08min	✗	37s
timeliness (B)	∞	—	✓	37:34min	✓	12:49min	✓	34:49h		
fairness (A)	✗[1]	8:34min	✗[1]	31:06min	✗[2]	2:22min	✓[2]	14:11min	✓[2]	5:46h
			✓[2]	0:40h						
fairness (B)			✓	14:05h[2]			✓[2]	43:52h [3]		

[1] weak notion of contract [2] strong notion of contract [3] add. helping lemma (verified in 2:38min)

$$P_A(a,b,ttp,t) = \text{new } k;\ \text{event } Start_A(a,k);\ \text{out}(\text{`c'}, m_1); \left(\begin{array}{l}\text{in}(\text{`c'}, m_2);\ \text{out}(\text{`c'}, m_3);\\ \text{event } Contract_A(a,b,t,k)\end{array}\right) + \left(\begin{array}{l}\text{in}(\text{`r'}, r_2);\\ \text{event } Contract_A(a,b,t,k)\end{array}\right)$$

$$P_B(a,b,ttp) = \text{in}(\text{`c'}, m_1);\ \text{new } sess;\ \text{event } Start_b(b,sess);\ \text{out}(\text{`c'}, m_2); \left(\begin{array}{l}\text{in}(\text{`c'}, m_3);\\ \text{event } Contract_b(a,b,t,sess)\end{array}\right) + \left(\begin{array}{l}\text{out}(\text{`r'}, r_1); \text{in}(\text{`r'}, r_3);\\ \text{event } Contract_b(a,b,t,sess);\end{array}\right)$$

$$P_T = \text{in}(\text{`r'}, r_1); \text{out}(\text{`r'}, r_2); \text{out}(\text{`r'}, r_3);$$

Figure 7: Description of the toy fair non-repudiation protocol

Modelling Fairness. In this section, we will discuss our formulation of the fairness property. Intuitively, fairness may be expressed as follows:

"Either both parties can receive a contract or none of them can."

Suppose that C_A and C_B are logical formulas that represent the statement *"if A (respectively B) proceeds, she will receive the expected contract"*. This can indeed be expressed using the $Contract$ events that annotate the processes (see Figure 7). Then the above intuitive formulation can be expressed as

$$(C_A \wedge C_B) \vee \neg(C_A \vee C_B)$$
$$\Leftrightarrow (C_A \rightarrow C_B) \wedge (C_B \rightarrow C_A).$$

The second equivalent formulation expresses both fairness for A (first disjunct) and fairness for B (second disjunct). In our complete model, we consider the cases where A or B may be dishonest (modelled through corruption described below). Suppose that D_B expresses that B has been corrupted. In that case we do not require fairness to hold for B, but only for A. As in our calculus, protocol events can only be emitted by honest runs of the protocol, the attacker may not be able to raise the event $Contract_B$ for a corrupted B. Therefore, we model a fourth entity, a judge, which emits an event if enough evidence has been brought forward to prove that a contract was made:

$$P_J = (\text{in}(\text{`c'}, m_1);\ \text{event } Contract_{judge}(A,B,T)) \mid \cdots \mid (\text{in}(\text{`c'}, m_n);\ \text{event } Contract_{judge}(A,B,T))$$

where $m_1, \ldots, m_n$ are the messages that suffice as evidence of a contract. We assume the public variable T is part of $m_1, \ldots, m_n$ and describes the contract text. Suppose that J expresses that *"it is possible to raise event $Contract_{judge}$"*. Then, in the case where B is corrupted, C_B should be replaced by J, and fairness for A expressed as $J \rightarrow C_A$. Note that the judge is different from the TTP in particular, the judge never emits messages, but just an event if sufficient evidence for a contract was brought forward.

Following these ideas, and recalling that fairness is only required to hold for uncorrupted parties, we can express fairness for A in the first-order logic introduced in Section 5.

$$\forall i : temp, a, b, t : pub.Contract_{judge}(a,b,t)@i$$
$$\Rightarrow (\exists j : temp, k : msg.Contract_A(a,b,t,k)@j)$$
$$\vee (\exists k : temp.Corrupt(a)@k).$$

where $Corrupt$ is the event raised when a party has been corrupted. Fairness for B is obtained by switching A for B. Overall, fairness is the conjunction of these two conditions. Using our tool we show that fairness indeed holds for A.

Modelling timeliness. Timeliness guarantees that no honest participant is 'left hanging', i.e., stuck in a situation where it cannot continue without the help of another participant, while fairness guarantees that no honest party ends up without a contract if the other has one. Consider again the toy example. Even though fairness holds for A, once A has sent message m_1 he needs to wait for either m_2 or r_2. He does however not know whether one of these messages will ever arrive or if B simply stopped the protocol – A is left 'hanging'. This demonstrates that fairness does not imply timeliness, while the other direction is clear: even if a participant can always terminate, he might not always obtain the contract.

Timeliness expresses that a participant can always unilaterally (i.e. without the help of the other participant, but

possibly relying on the TTP) finish the protocol. Timeliness is modelled by annotating the processes with $Start$ events, expressing that a session has started (cf Figure 7) in addition to the $Contract$ events (and $Abort$ events in the more complex ASW and GJM protocols). The arguments of these events should identify the session. Then timeliness for A is expressed as

$$\forall i : temp, a, b, t, k. Start_A(a, k)@i \Rightarrow (\exists j : temp. Contract_A(a, b, t, k)@j) \vee (\exists j : temp. Corrupt(a)@j)$$

Again, no guarantee is required when A is corrupt and timeliness for B is formulated similarly. Using our tool we confirm that timeliness does not hold for A.

Modelling resilient channels and corruption. In our case studies, we found that it is very often not clear what exactly is required from resilient channels to achieve timeliness or fairness. Suppose Alice takes the role of the initiator in two sessions, and the role of the responder in another. Do all her sessions share the same channel, do the initiator sessions share the same channel, or does every session have a separate channel?

In the ASW protocol, we found that a reply from the TTP does not identify which responder session should receive it and we chose to model:

- For the toy and ASW protocol, one resilient channel to the TTP per participant and protocol role (either originator or responder), along with the corresponding return channel.
- For the GJM protocol, one channel per session, as this protocol does not carry any session information in its messages.

Our calculus provides only a single resilient channel, but the above assumptions can be trivially modelled via pattern matching. While the assumption of a channel per participant is standard (e.g., it is necessary for fairness of the Zhou-Gollman protocol [28]), the separation by protocol role is unusual. It is justified in the case of the ASW protocol, as a participant A that has two sessions with itself and aborts the protocol in both, might receive one of the two abort messages from session one in the other session. While this does not necessarily imply an attack, it makes it much more difficult to prove timeliness for our tool, while it only amplifies the assumption.

The corruption process raises an event to mark a party corrupted, and reveals its secret key to the adversary. Additionally, for each corrupted party we add a process that inputs any messages sent over resilient channels to these parties. This is important as any trace with undelivered messages is ignored and attacks might be missed.

$$! \text{ in}(\text{'c'}, \langle \text{'cor'}, x\rangle);\ \text{event } Corrupt(x);\ \text{out}(\text{'c'}, sk(x)); (! \text{ in}(\text{'r'}, \langle \text{'resp'}, x, m\rangle \ \mid\ ! \text{ in}(\text{'r'}, \langle \text{'orig'}, x, m\rangle))$$

8.2. ASW protocol

The optimistic contract-signing protocol by Asokan, Shoup, and Waidner [3] proceeds as follows. For a contract text T, the originator A sends a signature for T and a commitment to a freshly drawn nonce n_a in the form of a hash. The responder B confirms by signing this message and a commitment on another freshly drawn nonce, n_b. Both parties then exchange their nonces. (Note that we have left out the identifiers of originator, responder and TTP in the first message.) In case that A or B are not receiving a

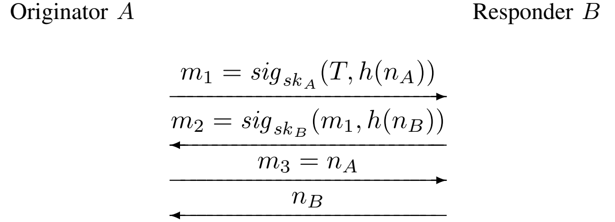
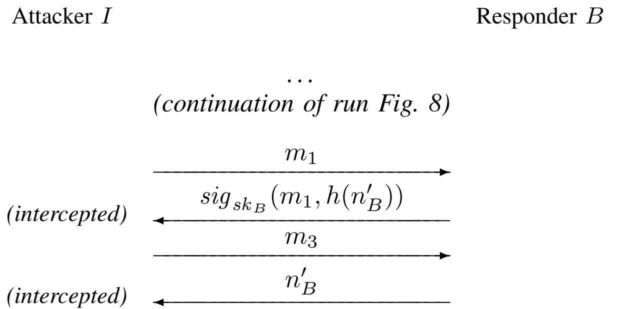


Figure 8: ASW protocol: honest protocol run

response in appropriate time, A may decide to abort the protocol (if the second message does not arrive), to resolve the contract with the TTP (if the fourth message does not arrive), or B may decide to resolve the contract (if the third message does not arrive). For brevity, we will only outline the parts of the corresponding abort/resolve-protocols when they are relevant to attacks below.

It is important to note that here *the complete transcript from the first to the fourth message constitutes the contract text*, including the nonces. As indicated by the original authors [3, Definition 3.1], each transcript of the honest protocol run identifies a different contract. In case the TTP is called, a second form of a valid contract is recognized, which consists of the TTP's signature on the first and the second message of the opportunistic protocol run, $sig_{sk_{TTP}}(m_1, m_2)$ for m_1 and m_2 of the form in Figure 8.

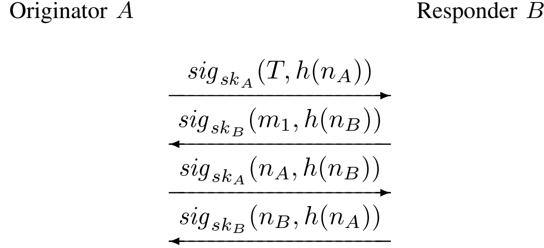
With this notion of contract, the following replay attack permits an attacker to create an arbitrary amount of different copies with the same contract text T for A and B, without A having any knowledge of this, nor A having any evidence that this attack took place, as the TTP is never contacted. Suppose the attacker observes the honest run above, he can commit to another contract with the responder B in the name of A, just by replaying the first and third message:



ASW protocol: Shmatikov/Mitchell attack.

This attack was discovered in a finite model by Shmatikov and Mitchell. The weakness here is that the third message is

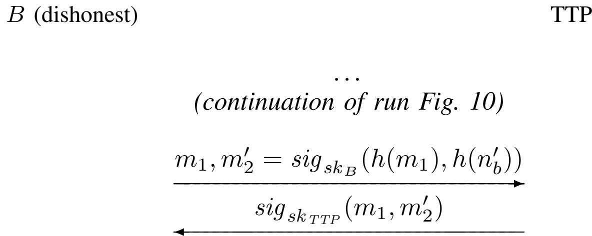
not related to the second message in any way, so Shmatikov and Mitchell proposed the following fix for the protocol.



ASW protocol: repaired version by Shmatikov and Mitchell.

Nevertheless we were able to show timeliness for A. We were unable to show timeliness for B, as the response of the TTP to a resolve request from B does not contain enough information to identify B's session. Rather than strengthening the assumption on the channel to avoid this possible confusion, we chose to modify the protocol by adding $h(n_B)$ to the response. Note that *a*) we did not find an actual attack, as a message that arrives in the wrong session would prevent this session to pick up any other messages, hence we conjecture that, overall, each message will be successfully picked up. Yet, the additional proof effort did not seem to justify the slight gain in generality. Furthermore, *b*) for the GJM protocol, we did avoid this problem by imposing a larger number of secret channels, as there was no hope for a similar fix in that protocol. For the protocol containing our and Shmatikov/Mitchell's modification, we managed to show timeliness for both parties.

Using our tool, we also found that surprisingly simple attack on the fairness of this repaired protocol. Suppose a dishonest B has signed a contract with A and wants to have a second copy of it. B can obtain a second copy without A's consent by calling the TTP to resolve with a 'refreshed' m_2, where n_B is substituted by a freshly drawn nonce n'_B.



ASW protocol: new attack.

If we alter the judge process, so that it identifies a contract with the text committed to, and the two signers, but not the nonce n_a, then we are able to show fairness for both parties. We call this property *fairness for the weaker notion of a contract*. Note that this rules out certain kinds of contract, e.g., if A and B exchange IOUs, one would expect each new IOU, even if it contains the same text, to correspond to a different contract, e.g., that three contracts saying 'A owes B $50' would amount to a debt of $150.

TABLE 2: OPC UA Secure Conversation results

Property	Time	Proof steps
all messages are received	1s	15
all messages were sent	7s	138
message order is respected	26s	204

8.3. GJM protocol

The fairness of the optimistic contract-signing protocol by Garay, Jakobsson, and MacKenzie (GJM) [16] was already analysed in previous work, but only in a bounded model. Under the assumption that each party has a reliable channel to and from the TTP for each session, we can automatically show timeliness for A and B. The verification proceeds automatically and without any additional lemma.

However, we immediately found an attack on fairness for A, even for the weak notion of contract. The optimistic protocol run, as well as recovery conducted by the trusted third party, will return a contract of the same form, namely

$$(sig(\langle `1', t\rangle, sk_A), sig(\langle `2', t\rangle, sk_B)),$$

where t is the contract set, and '$1'$ and '$2'$ just serve to distinguish these messages in a protocol run. Note that neither signature contains the identity of the respective contract partner. Hence it is easy, e.g., for a party X with a bad reputation, to obtain a contract A would only want to sign with B, just by replacing the second signature:

$$(sig(\langle `1', t\rangle, sk_A), sig(\langle `2', t\rangle, sk_X)).$$

This attack only applies if the contract text does not explicitly mention the signing parties but rather depends on the signers, e.g., "the signers agree to ...". If we require t to be of the form $\langle A, B, t'\rangle$, i.e., to contain the signing parties, we can automatically show fairness (for the weak notion of contract) for A and B. The protocol we show secure actually enjoys a small improvement: Garay et.al. assume the reliable channel to the TTP to additionally be secret. We lift this assumption, as only the responder's resolve message needs to be kept secret. Thus, we use asymmetric encryption in the transmission of this message, while the originator's resolve message and the abort message can remain unencrypted.

8.4. OPC UA Secure Conversation Protocol

To show that our approach can also be used beyond contract-signing, we analyzed the Secure Conversation Protocol, which is part of the United Architecture (UA) standard [24] developed by the OPC Foundation. The protocol implements a security layer designed for the use in industrial control systems, and aims at securing the data flow between two devices that share symmetric keys. It uses symmetric encryption and message authentication codes (MACs), and relies on sequence numbers to ensure the correct order of messages.

In the context of industrial control systems, the integrity of the data exchanged between two devices is extremely important. Modifying, injecting, or just reordering command

messages, e.g., in critical infrastructure such as the power grid, can have catastrophic effects by putting the system in a state beyond its safe operation limits.

Similar to fair exchange protocols, the protocol relies on a resilient channel to ensure message delivery, yet the protocol still needs to make sure that messages cannot be injected, duplicated or reordered. Using our approach we were able to show that in the OPC UA Secure Communication protocol, all messages are received only once and in the correct order. More precisely, we prove the following properties.

- All sent messages are received:

$$\forall i : temp, A, B, t : msg.\ Send(A, B, t)@i \Rightarrow (\exists j \gtrdot i : temp.\ Recv(A, B, t)@j)$$

- All received messages were sent before and are only received once:

$$\forall i : temp, A, B, t : msg.\ Recv(A, B, t)@i \Rightarrow (\exists j \lessdot i : temp.\ Send(A, B, t)@j \wedge \neg(\exists k \neq i : temp, A_2, B_2 : msg.\ Recv(A_2, B_2, t)@k))$$

- Any two messages that are received in a certain order were sent in that order:

$$\forall i, j : temp, A, B, m, n : msg.\ Recv(A, B, m)@i \wedge Recv(A, B, n)@j \wedge i \lessdot j \Rightarrow (\exists k, l : temp.\ Send(A, B, m)@k \wedge Send(A, B, n)@l \wedge k \lessdot l)$$

9. Conclusion

In this paper, we have presented a novel methodology for reasoning about liveness properties of cryptographic protocols in a machine-assisted manner without imposing artificial constraints on the size of protocol descriptions and executions as commonly done in prior work. Our findings from applying this methodology to the widely investigated class of fair exchange protocols notably demonstrate that such finiteness constraints do not constitute a purely academic limitation, but that they are responsible for not detecting actual weaknesses in such protocols.

Moreover, our approach of augmenting a higher-level calculus with key concepts for stating and reasoning about liveness properties and of subsequently designing a provably sound and complete translation into the widely used model of multiset rewriting allowed us to build upon recent advances in the automated verification of cryptographic protocols. In particular, we strongly benefit from the large degree of automation in the state-of-the-art verification tool Tamarin, and enable reasoning about liveness properties in Tamarin in a comprehensive manner through our translation.

10. Acknowledgements

This work was supported by the German Federal Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA) and EC SPRIDE, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC), as well as the Centre National de la Recherche Scientifique (CNRS) through funding for the PEPS projects ASSI and VESPA.

Appendix

Given a position p_0 and a set of positions P we denote by $p_0 \cdot P$ the set of positions $\{ p_0 \cdot p \mid p \in P \}$.

Definition 13. *Given a ground process P we define the function next* $\mathsf{next}(P)$ *as*

$$\mathsf{next}(P) \hat{=} \begin{cases} \emptyset & \text{if } P = 0 \\ (1 \cdot \mathsf{next}(P_1)) & \text{if } P = P_1 + P_2 \wedge \mathsf{blck}(P_1) \\ \cup (2 \cdot \mathsf{next}(P_2)) & \quad \wedge\ \mathsf{blck}(P_2) \\ (1 \cdot \mathsf{next}(P_1)) & \text{if } P = P_1 + P_2 \wedge \mathsf{blck}(P_1) \\ \cup \{ 2 \} & \quad \wedge \neg\mathsf{blck}(P_2) \\ \{ 1 \} & \text{if } P = P_1 + P_2 \wedge \neg\mathsf{blck}(P_1) \\ \cup (2 \cdot \mathsf{next}(P_2)) & \quad \wedge \mathsf{blck}(P_2) \\ \mathsf{children}(P) & \text{otherwise} \end{cases}$$

Next, we define the starting points of local progress. Intuitively, the set $\mathsf{From}(P)$ is the set of positions of P from where a progression starts.

Definition 14. *Given a ground process P we define the set* $\mathsf{From}(P) \hat{=} \mathit{from}(P, \top)$ *where*

$$\mathit{from}(P, b) \hat{=} \begin{cases} \{ [] \} \cup \bigcup_{p \in \mathsf{next}(P)} p \cdot \mathit{from}(P|_p, \mathsf{blck}(P)) \\ \quad \text{if } \neg\mathsf{blck}(P) \wedge b \\ \bigcup_{p \in \mathsf{next}(P)} p \cdot \mathit{from}(P|_p, \mathsf{blck}(P)) \\ \quad \text{otherwise} \end{cases}$$

We extend the previous notation: given a position p_0 and a set of sets of positions $\mathcal{P}$, $p_0 \cdot \mathcal{P}$ denotes $\{ p_0 \cdot P \mid P \in \mathcal{P} \}$.

Definition 15. *Given a process P we define the progression function $\pi\colon \mathsf{From}(P) \rightarrow 2^{2^{pos(P)}}$, as $\pi(p) = p.f(P|_p)$, where f is defined inductively on the structure of P:*

$$f(P) \hat{=} \begin{cases} \{ \{ [] \} \} & \text{if } \mathsf{blck}(P) \\ 1.f(P_1) \cup 2.f(P_2) & \text{if } P = P_1 \parallel P_2 \\ \{ \bigcup_{p \in \mathsf{next}^0(P)} S_p \mid S_p \in p.f(P|_p) \}, & \text{otherwise,} \end{cases}$$

where $\mathsf{next}^0(P)$ *is defined just like* $\mathsf{next}(P)$ *(i.e., replacing any occurence of* next *in Definition 13 by* next^0*), except that* $\mathsf{next}^0(0) = \{ [] \}$.

Using the progress function we relate positions in $\mathsf{From}(P)$ to positions they move to.

Definition 16. *Let P be a process and π its progress function. We define the binary relation $\mathcal{R}_\pi$ as*

$$(p, q) \in \mathcal{R}_\pi \text{ iff } p \in \mathsf{From}(P) \text{ and } \exists A. A \in \pi(p) \text{ and } q \in A.$$

We define the set $\mathsf{To}(P)$ *to be the range of $\mathcal{R}_\pi$.*

References

[1] Martín Abadi and Cédric Fournet. "Mobile Values, New Names, and Secure Communication". In: *28th ACM Symp. on Principles of Programming Languages (POPL'01)*. ACM, 2001, pp. 104–115.

[2] Alessandro Armando et al. "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications." In: *17th International Conference on Computer Aided Verification (CAV'05)*. LNCS. Springer, 2005, pp. 281–285.

[3] N. Asokan, Victor Shoup, and Michael Waidner. "Asynchronous protocols for optimistic fair exchange". In: *IEEE Symposium on Security and Privacy (S&P'98)*. IEEE Comp. Soc., 1998, pp. 86–99.

[4] Michael Backes et al. *A Novel Approach for Reasoning about Liveness...* Extended version, https://hal.inria.fr/hal-01396282.

[5] David A. Basin, Jannik Dreier, and Ralf Sasse. "Automated Symbolic Proofs of Observational Equivalence". In: *22nd Conference on Computer and Communications Security (CCS'15)*. ACM, 2015, pp. 1144–1155.

[6] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated Verification of Selected Equivalences for Security Protocols". In: *Symposium on Logic in Computer Science (LICS'05)*. IEEE Comp. Soc., 2005, pp. 331–340.

[7] Jan Cederquist and Muhammad Torabi Dashti. "An intruder model for verifying liveness in security protocols". In: *ACM Workshop on Formal methods in security engineering, (FMSE'06)*. 2006, pp. 23–32.

[8] Rohit Chadha, Max Kanovich, and Andre Scedrov. "Inductive methods and contract-signing protocols". In: *8th ACM Conference on Computer and Communications Security*. ACM, 2001, pp. 176–185.

[9] Rohit Chadha, Steve Kremer, and Andre Scedrov. "Formal Analysis of Multi-Party Contract Signing". In: *Journal of Automated Reasoning* 36.1-2 (2006), pp. 39–83.

[10] Rohit Chadha et al. "Automated verification of equivalence properties of cryptographic protocols". In: *ACM Transactions on Computational Logic* 17.4 (2016).

[11] Rohit Chadha et al. "Contract signing, optimism, and advantage". In: *CONCUR 2003 — Concurrency Theory*. LNCS. Springer-Verlag, 2003, pp. 366–382.

[12] Véronique Cortier, Graham Steel, and Cyrille Wiedling. "Revoke and Let Live: A Secure Key Revocation API for Cryptographic Devices". In: *19th ACM Conference on Computer and Communications Security (CCS'12)*. ACM, 2012, pp. 918–928.

[13] Cas J.F. Cremers. "The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols". In: *20th Conference on Computer Aided Verification (CAV'08)*. LNCS. Springer, 2008, pp. 414–418.

[14] Santiago Escobar, Catherine Meadows, and José Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V*. LNCS. Springer, 2009, pp. 1–50.

[15] Shimon Even and Yacov Yacobi. *Relations among Public Key Signature Systems*. Tech. rep. 175. Technion, 1980.

[16] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. "Abuse-Free Optimistic Contract Signing". In: *Advances in Cryptology—Crypto'99*. LNCS. Springer, 1999, pp. 449–466.

[17] Sigrid Gürgens and Carsten Rudolph. "Security analysis of efficient (Un-)fair non-repudiation protocols". In: *Formal Asp. Comput.* 17.3 (2005), pp. 260–276.

[18] Steve Kremer and Robert Künnemann. "Automated Analysis of Security Protocols with Global State". In: *35th IEEE Symposium on Security and Privacy (S&P'14)*. IEEE Comp. Soc., 2014, pp. 163–178.

[19] Steve Kremer and Robert Künnemann. "Automated analysis of security protocols with global state". In: *Journal of Computer Security* 24.5 (2016), pp. 583–616.

[20] Steve Kremer, Olivier Markowitch, and Jianying Zhou. "An Intensive Survey of Fair Non-repudiation Protocols". In: *Computer Communications* 25.17 (2002), pp. 1606–1621.

[21] Steve Kremer and Jean-François Raskin. "A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols". In: *Journal of Computer Security* 11.3 (2003), pp. 399–429.

[22] Steve Kremer and Jean-François Raskin. "Game Analysis of Abuse-Free Contract Signing". In: *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*. IEEE Comp. Soc., 2002, pp. 206–220.

[23] Leslie Lamport. "Proving the correctness of multiprocess programs". In: *IEEE transactions on software engineering* 2 (1977), pp. 125–143.

[24] *OPC Unified Architecture Specification, Part 6: Mappings, Release 1.02*. OPC Foundation. 2012.

[25] Bruno Blanchet, Ben Smyth, and Vincent Cheval. *ProVerif 1.88: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*. 2013.

[26] Benedikt Schmidt et al. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *25th IEEE Computer Security Foundations Symposium (CSF'12)*. IEEE Comp. Soc., 2012, pp. 78–94.

[27] Benedikt Schmidt et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *25th International Conference on Computer Aided Verification (CAV'13)*. LNCS. Springer, 2013, pp. 696–701.

[28] Steve Schneider. "Formal Analysis of a Non-Repudiation Protocol". In: *11th IEEE Computer Security Foundations Workshop (CSFW'98)*. 1998, pp. 54–65.

[29] Vitaly Shmatikov and John C. Mitchell. "Finite-state analysis of two contract signing protocols." In: *Theor. Comput. Sci.* 283.2 (2002), pp. 419–450.