Tidy: Symbolic Verification of Timed Cryptographic Protocols

Gilles Barthe
gjbarthe@gmail.com
MPI-SP & IMDEA Software Institute
Bochum, Germany

Ugo Dal Lago
ugo.dallago@unibo.it
University of Bologna & INRIA Sophia Antipolis
Bologna, Italy

Giulio Malavolta
giulio.malavolta@mpi-sp.org
MPI-SP
Bochum, Germany

Itsaka Rakotonirina
itsaka.rakotonirina@mpi-sp.org
MPI-SP
Bochum, Germany

ABSTRACT

Timed cryptography refers to cryptographic primitives designed to meet their security goals only for a short (polynomial) amount of time. Popular examples include timed commitments and verifiable delay functions. Such primitives are commonly used to guarantee fairness in multiparty protocols ("either none or all parties obtain the output of the protocol") without relying on any trusted party. Despite their recent surge in popularity, timed cryptographic protocols remain out of scope of current symbolic verification tools, which idealise cryptographic primitives as algebraic operations, and thus do not consider fine-grained notions of time.

In this paper, we develop, implement, and evaluate a symbolic approach for reasoning about protocols built from timed cryptographic primitives. First, we introduce a timed extension of the applied π-calculus, a common formalism to specify cryptographic protocols. Then, we develop a logic for timed hyperproperties capturing many properties of interest, such as timeliness or time-limited indistinguishability. We exemplify the usefulness of our approach by modelling a variety of cryptographic protocols, such as distributed randomness generation, sealed-bid auctions, and contract signing.

We also study the decidability of timed security properties. On the theoretical side, we reduce the decision of hyperproperties expressed in our logic to a form of constraint solving generalising standard notions in protocol analysis, and showcase the higher complexity of the problem compared to similar well-established logics through complexity lower bounds. On the automation side, we mechanise proofs of timed safety properties by relying on the Tamarin tool as a backend, a popular symbolic protocol analyser, and validate several examples with our approach.

CCS CONCEPTS

• **Security and privacy → Logic and verification**; **Formal security models**.

KEYWORDS

Security Protocols; Timed Cryptography; Symbolic Models

CCS '22, November 7–11, 2022, Los Angeles, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3559343

ACM Reference Format:
Gilles Barthe, Ugo Dal Lago, Giulio Malavolta, and Itsaka Rakotonirina. 2022. Tidy: Symbolic Verification of Timed Cryptographic Protocols. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3559343

1 INTRODUCTION

Multi-party protocols are distributed programs executed by heterogeneous entities, with the underlying goal of computing an output, e.g., a signed contract, the result of an auction, or simply a random value. Although the output is intended to be shared by all participants, some parties may have an incentive to prevent other parties from obtaining it. Or, they may decide to abort the computation if the potential output is not of their convenience. To exemplify the issue, consider the prototypical scenario where two mutually distrustful parties (A, B) want to agree on a common random string. A natural protocol to achieve this goal would be to let A sample a random string x locally and then send a *commitment* on x to B. After B responds with a uniformly sampled bit y, A can then reveal the committed message and the output of the protocol is set to $z = x \oplus y$. The rationale of this protocol is that y cannot depend on x (the commitment hides x to B), while at the same time x cannot be changed "after the fact" (the commitment binds A to x).

Unfortunately, this simplistic analysis neglects to account for the fact that A learns the output of the protocol before B. Therefore, A may decide to stall the protocol, by not opening the commitment, if the output is not favourable. Using this strategy, A can bias the distribution of z arbitrarily. The root of the problem is that this protocol does not satisfy *fairness*, a cryptographic notion that guarantees that either all parties obtain the output of the protocol, or none of them does. This property is notoriously hard to enforce, and it is in fact known to be unachievable in classic models of asynchronous communications [28, 33]. One way to circumvent this impossibility result is to assume a *trusted third party* (TTP), responsible to deliver the output to all participants. However, the assumption of a TTP is unrealistic in many scenarios, raising the question of whether multi-party protocols can realise fairness without trusted parties.

Fairness via Timed Cryptography. Modern cryptography bypasses this barrier by relying on *timed cryptographic primitives*, i.e., primitives that are assumed not to be computable in less than a chosen amount of time [53]. As an example, *timed* commitments [18] allow to hide a message for a pre-determined amount of time d.

Going back to our above problem of string sampling, timed commitments are the right tool to construct a fair string sampling protocol: A can no longer withhold the opening of its commitment, since it will anyway be public in time d. On the other hand, the timed commitment still ensures that B does not learn the committed value during the protocol, as long as the protocol terminates in time less than d. Importantly, both parties need to place a strict timeout on the duration of the protocol to ensure that this requirement is met.

Other examples of timed cryptographic primitives include verifiable delay functions [17], verifiable timed signatures [57], and (homomorphic) time-lock puzzles [49, 53]. These primitives have been developed to enforce fairness at a cryptographic level in a variety of settings, and count numerous applications: sealed-bid auctions [18], fair contract signing [18], randomness beacons [17], computational time-stamping [17], or even electronic voting [49].

Symbolic Analysis of Cryptographic Protocols. Cryptographic protocols are complex and error-prone, but there exist symbolic models in which these protocols can be specified and verified. These models, such as the Dolev-Yao model [31] and the subsequent applied π-calculus [1], view cryptographic primitives as (idealised) algebraic constructions, and focus on the interactions of the protocol's participants. Thanks to this abstraction, security properties can be expressed as logical formulae and can be verified, expectedly automatically, by means of decision procedures or proof systems. The symbolic analysis of cryptographic protocols is a flourishing area of research, supported by performing tools that have been used to analyse or guide the design of widely deployed protocols [9]. However, current symbolic models disregard low-level considerations such as the execution time and, as such, are not suitable to reason about protocols built on top of timed cryptography.

Contributions

In this paper, we propose the first symbolic model of cryptographic protocols built over timed cryptography.

(1) We propose a model *à la Dolev-Yao* of timed cryptography, and of timed distributed processes executed in an unbounded adversarial environment. For that we extend the applied π-calculus [1], a classical framework for specifying cryptographic protocols, with time constraints.
(2) We introduce a logic for expressing security properties of such processes, inspired by popular logics for hyperproperties such as HyperCTL* [26, 39]. We call ours Hyper*tidy* CTL* as it is designed to express hyperproperties in *timed Dolev-Yao* models. A major difference with existing hyperlogics—designed for finite-state systems—is our ability to quantify over unbounded (adversarial) computations. This makes our framework substantially more expressive in the context of protocols, but also more complex from a decidability standpoint.
(3) We illustrate this gap through complexity lower bounds for the verification problem. Typically, even the non-relational fragment of our logic, *tidy* CTL*, is undecidable while the analogue problem in CTL* (i.e., model checking) is **PSPACE** complete. On the positive side, we show that verifying that a bounded number of sessions of a protocol satisfy a Hyper*tidy* CTL* formula can be reduced to a form of *constraint solving*. It is inspired by popular analogues in protocol analysis used to verify reachability and indistinguishability properties in the untimed case [22, 23].
(4) Finally, we verify timed safety properties (*tidy* LTL) by relying on Tamarin, a popular automated prover for (untimed) security protocols. We develop an approach to handle time constraints while relying on the tool as a backend soundly, and successfully analyse several properties of interest of various protocols.

For space reasons, some technical contributions and proofs have been omitted and can be found in a technical report, available at [10] together with experimental files.

2 MOTIVATING EXAMPLE

Timed commitments [18] are a classic tool to build fair protocols. In this section, we give a first insight of our model by outlining the construction and verification of the fair sampling protocol briefly described in the introduction [18].

Recall that commitments are cryptographic primitives allowing a party to commit to a value while keeping it hidden from other parties, until a moment of their choice. Timed commitments enhance this scheme with an additional algorithm for revealing the committed value within some time d, even if the committer refuses to open its commitment. We model them by *function symbols*, written:

$$\mathsf{commit}[x,y,d] \qquad \mathsf{open}[x,y,z] \qquad \mathsf{force}[x] \qquad \mathsf{valid}[x,d]$$

Variables named as $x, y, z, \ldots$ stand for arguments serving as cryptographic material, while d, t stand for durations and time parameters. The expression $\mathsf{commit}(m,r,d)$ for example represents a commitment of the message m that can be opened with the opening data r, or forced in time d. The opening of a commitment c with m, r is represented by $\mathsf{open}(c,m,r)$, whereas $\mathsf{force}(c)$ is a forced recovery of m from c. Finally, $\mathsf{valid}(m,d)$ tests that the bitstring m is a well-formed commitment of time parameter d. We express the operational behaviour of these expressions by *timed rewriting rules*:

$$\begin{aligned} \mathsf{open}(\mathsf{commit}(x,y,d),x,y) &\to \mathsf{ok} \\ \mathsf{valid}(\mathsf{commit}(x,y,d),d) &\to \mathsf{ok} \\ \mathsf{force}(\mathsf{commit}(x,y,d)) &\xrightarrow{+d} x \end{aligned}$$

The first rule models the correctness of commitments: opening $\mathsf{commit}(x,y,d)$ with the correct data instantaneously yields validation. The second rule simply states that one can test that a message is valid commitment parametrised with d (without opening it). Finally, the last rule models a forced opening: the message x can always be recovered in time d. This is indicated by the $+d$ label, whereas other rules are implicitly labelled $+0$. The absence of other rules models that no information can be extracted about the committed message without knowing the opening data, or spending d units of time forcing it. This scheme can be used in a string-sampling protocol, as the exclusive or (xor, $\oplus$) of two bitstrings x and y held by parties A and B, respectively. Its goal is to ensure that no party can bias the outcome, i.e., $x \oplus y$ is uniformly distributed if at least A or B follows the protocol and samples its bitstring uniformly:

$$\begin{aligned} A \to B &: \ \mathsf{commit}(x,r,d) \\ B \to A &: \ y \\ A \to B &: \ x, r \end{aligned}$$

After this exchange, A and B can compute $x \oplus y$. However, to ensure that the result is unbiased, several additional precautions have to be taken. We formalise in our model the views of each party below.

View of A. First, A generates a fresh nonce r and outputs the commitment $\mathsf{commit}(x, r, d)$ at time t. Then, it waits for a response y, but only accepts it if received at some time $t' < t + d$. This way, B is too short on time to force the commitment, and can therefore not compute y depending on the retrieved value of x. If A received y in time, it accepts $x \oplus y$ as the protocol's result and reveals x and r. We express this process in a timed extension of applied π-calculus:

$$\begin{aligned} A(x,d) = {} & \mathsf{new}\ r;\ \mathsf{out}(\mathsf{commit}(x,r,d))@t; \\ & \mathsf{event}\ \mathsf{Standby}@t_s\ \mathsf{when}\ t_s < t+d; \\ & \mathsf{in}(y)@t'\ \mathsf{when}\ t' < t+d; \\ & \mathsf{event}\ \mathsf{Accept}(x \oplus y); \mathsf{out}(x); \mathsf{out}(r); 0. \end{aligned}$$

Inputs and outputs are materialised by $\mathsf{in}(\cdot)$ and $\mathsf{out}(\cdot)$ instructions, while 0 simply indicates a terminated process. Timestamps and time conditions are expressed using the @ and when operators. Finally, the events (Standby and Accept) are not actual instructions of the protocol: they should rather be seen as meta-events identifying specific phases of the protocol, for formalising security properties. Here for example, Standby indicates that A is still waiting for B's reply at time t_s. The expected security property is then, informally:

> *Fairness towards A: the bitstring x emitted by A remains secret until B responds (if it does).*

Note that *timed* commitment are only beneficial to B, i.e., the property would hold for a protocol relying on a regular commitment. We formalise the property by requiring that no computation X of the adversary (here B) may result in x while A still awaits an answer. We express this in our model by the following *tidy CTL* formula*:

$$\forall \pi . \forall z . \mathsf{F}\ \mathsf{Accept}(z)_\pi \Rightarrow \neg \mathsf{K}(x)_\pi\ \mathsf{U}\ \mathsf{Standby}_\pi$$

The syntactic sugar $\mathsf{K}(x)_\pi \triangleq \exists X, X \vdash_\pi x$ is a formula intuitively expressing that x is computable with access to the outputs sent by A during an execution π. The operators F and U are respectively read as "finally" and "until" and the property itself thus intuitively means "*for all executions π of $A(x,d)$ that eventually accepts a result z, the attacker B cannot compute x before the event* Standby". Note in particular how the quantification $\forall \pi$ requires to consider executions where the Standby event occurs arbitrarily close to the reception of y. We also later formalise stronger versions of this property exploiting the full expressivity of our logic for hyperproperties.

View of B. Dually, B awaits for a value x_c, and upon reception checks that it is a valid commitment. It then awaits for x and r, and attempts to open x_c with these values. If everything proceeds as expected, B accepts the value $x \oplus y$; otherwise, it forces the commitment through process $F = \mathsf{event}\ \mathsf{Accept}(\mathsf{force}(x_c) \oplus y); 0$.

$$\begin{aligned} B(y,d) = {} & \mathsf{in}(x_c);\ \mathsf{if}\ \mathsf{valid}(x_c,d) = \mathsf{ok}\ \mathsf{then} \\ & \quad \mathsf{out}(y);\ \mathsf{event}\ \mathsf{Standby}; (F + \mathsf{in}(x); (F + \mathsf{in}(r); C)) \\ C = {} & \mathsf{if}\ \mathsf{open}(x_c, x, r) = \mathsf{ok}\ \mathsf{then}\ \mathsf{event}\ \mathsf{Accept}(x \oplus y); 0\ \mathsf{else}\ F \end{aligned}$$

A sum $P + Q$ indicates an (external) non deterministic choice, i.e., B may execute either of two process options P or Q. In particular, B always has the choice to force the commitment instead of waiting for further opening data. The desired security property is then:

> *Fairness towards B: after B sent its bitstring y, the protocol can always proceed until the end, even if A aborts.*

A classical approach to formalise such *liveness* properties is to assume that B always executes instructions until being stuck to wait for an input from A. In practice, the *tidy* CTL* formula

$$\mathsf{stuck} = \forall \pi .\ \tau_\pi\ \mathsf{U}^?\ \exists X .\ \mathsf{in}(X)_\pi$$

captures this idea that B cannot progress. Indeed, τ_π can be understood here as "*the execution π is pending*", while $\varphi\ \mathsf{U}^?\ \psi$ ("weak until") is a relaxed variant of $\varphi\ \mathsf{U}\ \psi$ allowing φ to hold forever, should ψ never be true. This lets the stuck formula be read as "*the first action (if any) of any potential executions π starting from the current state is an input from A*". Therefore, the following formula:

$$\mathsf{Progress} = \mathsf{G}\ (\neg \mathsf{stuck} \Rightarrow \mathsf{F}\ \mathsf{stuck})$$

expresses that B always keeps going until being stuck, as $\mathsf{G}\ \varphi$ ("globally") means that φ is true all along the protocol execution. Altogether, fairness towards B is captured by the following formula:

$$\forall \pi .\ \mathsf{Progress} \Rightarrow \mathsf{G}\ (\mathsf{Standby}_\pi \Rightarrow \mathsf{F}\ \exists z .\ \mathsf{Accept}(z)_\pi)$$

The second part of the formula is read as "*at any point, if the event* Standby *occurs in the execution π, then* Accept(z) *will occur later*".

Rest of the paper. In the following sections, we formalise the timed extension of applied π-calculus used above. We also introduce a rich language for expressing security properties, Hyper*tidy* CTL*, generalising the simple (non-relational) formulas seen so far, and illustrate their use by modelling several protocols of interest in our framework. We then study the decidability of verifying that a protocol with a fixed number of sessions satisfies a Hyper*tidy* CTL* formula. We show that the problem is undecidable in general despite boundedness, but can be reduced to forms of constraint solving. Finally, we formalise a Tamarin-based approach to prove fairness towards A, as well as properties of other relevant protocols.

3 APPLIED π-CALCULUS WITH TIME

We first formalise our extension of the applied π-calculus including a support for timed cryptography.

3.1 Cryptographic Primitives and Messages

Term Algebra. We first review the algebraic setting we use to model cryptographic primitives and protocol messages. All the material is standard, except for the notions of time and cost, which are new. We start with *atomic values*:

$$\mathsf{A} = \mathcal{F}_0 \uplus \mathcal{N} \uplus \mathcal{X} \uplus \mathbb{R}^+ \qquad \text{and with } \mathcal{TX} \subseteq \mathcal{X}$$

The infinite set $\mathcal{F}_0$ of *constants* is used to model public values such as identities, or any value known to the adversary. On the contrary, the infinite set $\mathcal{N}$ of *names* models private values such as nonces or long-term keys. The infinite set $\mathcal{X}$ contains the *variables*, including a distinguished infinite subset $\mathcal{TX}$ of *temporal variables* used to specifically bind time values, represented by elements of $\mathbb{R}^+$.

In particular, the set of atomic values is implicitly typed: some values represent time, others cryptographic material. For the sake of readability, we gloss over this distinction and implicitly assume that all incoming notions (terms, substitutions, etc.) are well-typed. Next, we introduce a notion of *signature*, that is a finite set

$$\mathcal{F} = \{\mathsf{f}[x_1, \ldots, x_n],\ \mathsf{g}[x_1, \ldots, x_m],\ \mathsf{h}[x_1, \ldots, x_p], \ldots\}$$

used to represent operations such as cryptographic primitives. A symbol $\mathsf{f}[x_1, \ldots, x_n]$ models a primitive taking n arguments materialised by the variables x_i, the temporal variables among them indicating time parameters. A *cost* for f is then an arithmetic expression $cost(\mathsf{f})$ over temporal variables among $x_1, \ldots, x_n$. The cost of a function can be seen as a static amount of time necessary to compute it, regardless of its result. From all this, we can then define a standard notion of *term* to model computations.

Definition 3.1 (Term). A *term* is an atomic value $a \in \mathsf{A}$, or a function symbol f or an arithmetic operator $(+, \times, \ldots)$ applied to other terms while respecting arities and types. We write $\mathcal{T}(S)$ the set of terms built from the atomic values and functions of $S \subseteq \mathsf{A} \cup \mathcal{F}$.

Finally, the notion of *substitution* σ is defined as usual, i.e.,

$$\sigma = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$$

is a mapping where $x_1, \ldots, x_n$ are pairwise distinct variables forming the *domain* of σ written $dom(\sigma)$, and $u_1, \cdots, u_n$ are terms. The application of a substitution σ to a term t is denoted by $t\sigma$ which is, informally, the term obtained by simultaneously replacing all occurrences of x_i in t by $u_i = x_i\sigma$. Regarding notations, we write $\sigma \cup \sigma'$ the substitution of domain $dom(\sigma) \cup dom(\sigma')$ that extends both σ and σ' (provided they coincide on $dom(\sigma) \cap dom(\sigma')$); and $\sigma\sigma'$ refers to the composition $\sigma' \circ \sigma$, i.e., for all terms t, $t(\sigma\sigma') = (t\sigma)\sigma'$.

Rewriting. Terms have an operational behaviour (including the execution time) defined by a rewriting system that we augment with a cost function.

Definition 3.2 (Timed rewriting). A *(timed) rewriting system* $\mathcal{R}$ is a finite set of triples (ℓ, d, r), with ℓ, r terms that do not contain names, and d an arithmetic expression over temporal variables of ℓ. We call such triples (timed) *rewrite rules*, written

$$\ell \xrightarrow{+d} r\,.$$

We call d the *cost* of the rule (omitted when null). It intuitively indicates the execution time of the corresponding computation. In particular, a rewriting system $\mathcal{R}$ induces a rewriting relation $\xrightarrow{+d}_{\mathcal{R}}$ that is the closure of $\mathcal{R}$ under substitution and context. That is:

- $\ell\sigma \xrightarrow{+d\sigma} r\sigma$ for any $(\ell \xrightarrow{+d} r) \in \mathcal{R}$ and substitution σ;
- $C[u] \xrightarrow{+d} C[v]$ if $u \xrightarrow{+d} v$ and $C[\cdot]$ is a linear context, i.e., an application of function symbols on top of a term.

In practice, we only consider rewriting systems defining convergent computations for simplicity and coherence of our definitions:

Definition 3.3 (Normal form and Convergence). A term t is said to be in *normal form* if it cannot be reduced by $\rightarrow_{\mathcal{R}}$. A rewriting system is *convergent* if for all terms t, there exists a unique term in normal form, written $t\!\downarrow$, such that $t \xrightarrow{+d_1}_{\mathcal{R}} \cdots \xrightarrow{+d_n}_{\mathcal{R}} t\!\downarrow$.

Symbolic models also sometimes consider (unoriented) *equations* to specify algebraic properties such as associativity and commutativity, typically useful to model $\oplus$. As they would be untimed in our context and are irrelevant to all security properties we consider, we omit them from the model for the sake of succinctness.

3.2 Protocols

Syntax. We model distributed protocols through a notion of *(timed) process*, mostly borrowed from the applied π-calculus, except that instructions may have timestamps and time conditions. These conditions are typically built from classical arithmetic or logical operators such as $+$, $\times$, $\wedge$, or $<$, with their usual interpretation. Recalling the motivating example, an example of a time condition is $t' < t + d$ for some temporal variables $t, t', d \in \mathcal{TX}$. We also let a dedicated set of function symbols and constants $\mathcal{F}_e$, called *events*.

Definition 3.4 (Process). The grammar of *processes* is:

$$\begin{array}{lll}
P ::= & 0 & \textit{null process} \\
& \mathsf{new}\ k; P & \textit{new name} \\
& \mathsf{in}(u, x)@t\ \mathsf{when}\ b; P & \textit{timed input} \\
& \mathsf{out}(u, v)@t\ \mathsf{when}\ b; P & \textit{timed output} \\
& \mathsf{event}\ \mathsf{Ev}(\vec{u})@t\ \mathsf{when}\ b; P & \textit{timed event} \\
& P \mid P & \textit{parallel composition} \\
& !\,P & \textit{replication} \\
& P + P & \textit{external choice}
\end{array}$$

where $k \in \mathcal{N}$, u, v are terms, $\vec{u}$ is a sequence terms (respecting the number of arguments of the event symbol $\mathsf{Ev}[\vec{x}] \in \mathcal{F}_e$), $x \in \mathcal{X}$, $t \in \mathcal{TX}$, and b is a time condition (that may make reference to t). Naturally, for succinctness, the timestamp t and the time conditions b may be omitted if unused, thus subsuming the syntax of the usual applied π calculus for untimed processes.

Intuitively, new k binds a name k, typically modelling a fresh session nonce. An instruction $\alpha@t$ when $b; P$ executes α, provided b is satisfied, and records the current time inside t. More specifically, the instructions $\mathsf{in}(u, x); P$ and $\mathsf{out}(u, v); P$ model, respectively, inputs and outputs on a communication channel u. As the adversary is assumed to control the communication network, in case u is public (e.g., $u \in \mathcal{F}_0$), an output on u adds v to the adversary's knowledge while an input x is crafted by the adversary, possibly computing a new message from previous outputs. In case u is private (e.g., $u \in \mathcal{N}$), the communication is done synchronously (*internal communication*), without adversarial interferences. Most often, we will however use the simpler notations:

$$\mathsf{in}(x); P \qquad\qquad \mathsf{out}(v); P$$

referring to an implicit arbitrary public channel. Finally, event $\mathsf{Ev}(\vec{u})$ is a meta instruction used to formalise security properties. Regarding structural operators, $P \mid Q$ models two processes executed concurrently. Its unbounded analogue $!\,P$ represents infinitely many parallel copies of P. The *external* non-deterministic choice $P + Q$ is a classical extension of the calculus that executes either P or Q. The branching is only effective to actually execute an instruction from P or Q, thus preventing to branch to a stuck process; this makes it more suited for modelling liveness than some of its variants [7].

Syntax Extensions. One may note that our grammar does not provide a syntax for "if $u = v$ then P else Q" to perform tests as in the motivating example. It can however be encoded as the process

$$(\mathsf{event}\ \mathsf{Pos}(u, v); P) + (\mathsf{event}\ \mathsf{Neg}(u, v); Q)$$

$$\frac{\xi \vdash_{\Phi@t'} u \qquad t' \leqslant t}{\xi \vdash_{\Phi@t} u} \qquad \frac{a \in \mathsf{A}}{a \vdash_{\Phi@0} a} \qquad \frac{x@t \in dom(\Phi)}{x \vdash_{\Phi@t} x\Phi} \qquad \frac{\xi \vdash_{\Phi@t} u \qquad u \xrightarrow{+d}_{\mathcal{R}} v}{\xi \vdash_{\Phi@t+d} v}$$

$$\frac{\mathsf{f}[x_1, \dots, x_n] \in \mathcal{F} \qquad \forall i, \xi_i \vdash_{\Phi@t_i} u_i \qquad \sigma = \{x_i \mapsto u_i\}_{i=1}^{n}}{\mathsf{f}(\xi_1, \dots, \xi_n) \vdash_{\Phi@cost(\mathsf{f})\sigma+(\max_i t_i)} \mathsf{f}(u_1, \dots, u_n)}$$

Figure 1: Inference rules for dated deducibility

and by enforcing within security properties (see Section 4) that $\mathsf{Pos}(u, v)$ always implies that u and v have the same normal form, and $\mathsf{Neg}(u, v)$ implies that they do not. Such encodings of tests are standard in practical analysers, see, e.g., [43], and we will therefore use the "if . . . then . . . else" syntactic sugar for convenience. Another convenient syntax extension is the let binding let $x = u$ in P that evaluates the term u, and binds its normal form to x in P. This may be crucial, for example, to precompute the result of a timed primitive that will be used several times in P. It is encodable using internal communications, i.e., given a fresh name $e \in \mathcal{N}$:

$$\mathsf{out}(e, u); 0 \mid \mathsf{in}(e, x); P$$

3.3 Adversaries

We consider an adversarial semantics where processes are executed in parallel with an adversary that impersonates corrupted parties and builds knowledge from observing the values output by processes. We define in this section the related notions (adapted to the timed setting). First of all, we introduce a form of store recording the computations performed during the execution—including the adversarial knowledge. Formally, a *dated frame* is a substitution:

$$\Phi = \{x_1@t_1 \mapsto u_1, \dots, x_n@t_n \mapsto u_n\}$$

where an entry $x_i@t_i \mapsto u_i$ models a term u_i computed at time $t_i \in \mathbb{R}^+$ and stored under the handle $x_i \in \mathcal{X}$. This induces a notion of *dated computability*, where ξ, u are terms and Φ is a dated frame, defined by the inference rules of Figure 1. Intuitively, $\xi \vdash_{\Phi@t} u$ indicates that, by time t, the evaluation of $\xi\Phi$ has resulted in u.

Example 3.1. Consider the following dated frame giving access to a commitment and its opening data $m, r \in \mathcal{N}$:

$$\Phi = \{x_1@t_1 \mapsto \mathsf{commit}(m, r, d),\ x_2@t_2 \mapsto m,\ x_3@t_3 \mapsto r\}$$

The commitment may be opened using the data revealed at time t_2, t_3, or forced starting at time t_1 and for a duration of d. Formally:

$$\mathsf{open}(x_1, x_2, x_3) \vdash_{\Phi@\max(t_1,t_2,t_3)} \mathsf{ok} \qquad \mathsf{force}(x_1) \vdash_{\Phi@t_1+d} m \quad ◀$$

The adversarial computations themselves (with access to a dated frame) are then described by means of *recipes*.

Definition 3.5 (Recipe). We let an infinite set $\mathcal{AX} \subseteq \mathcal{X}$ of dedicated variables called *axioms*, not used in processes but possibly in a frame's domain. A *recipe* is then a term $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathbb{R}^+ \cup \mathcal{AX})$.

The axioms serve as handles to the protocols outputs, that the adversary reads by spying on the network. Thus, recipes describe how the adversary computes a message from the observations stored in a dated frame Φ. Yet recipes may only use axioms, and not arbitrary variables, as the latter identify internal computations of the process that the adversary has not access to. Names are prohibited in recipes since they model secret values.

3.4 Operational Semantics

The actual operational semantics of the calculus operates on an extended form of process recording time and attacker's knowledge.

Definition 3.6 (Extended process). An *extended process* is a tuple $(\mathcal{P}, \Phi, t)$, with $\mathcal{P}$ multiset of processes, Φ dated frame, and $t \in \mathbb{R}^+$. We often interpret a process P as the extended process $(\{\!\{P\}\!\}, \emptyset, 0)$.

Intuitively, $\mathcal{P}$ is the multiset of subprocesses currently being executed in parallel. The concrete time value t indicates the time elapsed since the beginning of the execution, while Φ records the adversary's knowledge and the values computed during the protocol. The semantics is then a labelled transition relation $\xrightarrow{\alpha}$, where α is called an *action* that may be either of:

1. an *input action* $\mathsf{in}(\xi, \zeta)$ modelling the reception of a message forged by the adversary, where ξ, ζ are recipes;
2. an *output action* $\mathsf{out}(\xi, \mathsf{ax})$ modelling a message sent on the network, and recorded in the frame under axiom ax;
3. a non observable action, namely either *silent action* τ, or an *event action* $\mathsf{Ev}(\vec{u})$, $\vec{u}$ sequence of terms in normal form.

We comment the relation, formalised in Figure 2. Rules (In) and (Out) tackle public communications: outputs increase the attacker's knowledge (i.e., add an axiom to $dom(\Phi)$), and inputs are computed using previous outputs (i.e., through a recipe ζ). Note that Rule (In) renames x as x' simply to avoid conflicts when storing the result in Φ. The communication in Rules (In) and (Out) is public in the sense that the channel u is computable by the adversary, using recipe ξ. On the contrary, no interactions with the adversary arise in (Comm). Rule (Event) triggers a meta action, materialising some control points to formalise security properties. Observe in particular that in the semantics of all instructions $\alpha@t$ when b, the time condition $b\{t \mapsto t_0\}$ is required to hold, which is how we handle that the time variable t may appear in b. Finally, (Par) and (Repl) spawn parallel threads, while (Choice) uses either of two processes for the next transition. Rule (Tic) then lets time elapse, structuring the temporal behaviour of the protocol. Process executions thus alternate between (Tic) rules (that make time progress) and other transitions (that require enough time to have elapsed).

Definition 3.7 (timed trace). A *timed trace* T of an extended process A_0 is an infinite sequence of transitions of Figure 2:

$$T : A_0 \xrightarrow{\alpha_1} A_1 \xrightarrow{\alpha_2} A_2 \xrightarrow{\alpha_3} \cdots$$

Finite portions of T may be more succinctly written $A_p \overset{w}{\Longrightarrow} A_q$ when the intermediary processes are unimportant, where w is the word $\alpha_{p+1} \cdots \alpha_q$ with τ removed. In addition, we require that T is *temporally structured*, i.e., writing $A_i = (\mathcal{P}_i, \Phi_i, t_i)$:

1. *time elapses between different actions*: $\forall i \in \mathbb{N},\ t_i < t_{i+2}$;

$$(\{\!\!\{\mathsf{in}(u,x)@t \text{ when } b;P\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\mathsf{in}(\xi,\zeta)} (\{\!\!\{P\rho\}\!\!\} \cup \mathcal{P}, \Phi \cup \{x'@t_0 \mapsto v\!\downarrow\}, t_0)$$
$$\text{with } \rho = \{t \mapsto t_0, x \mapsto x'\},\ x' \text{ fresh renaming of } x,\ b\rho \text{ holds},\ \xi \vdash_{\Phi@t_0} u\!\downarrow, \text{ and } \zeta \vdash_{\Phi@t_0} v\!\downarrow \qquad (\textsc{In})$$

$$(\{\!\!\{\mathsf{out}(u,v)@t \text{ when } b;P\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\mathsf{out}(\xi,\mathsf{ax})} (\{\!\!\{P\rho\}\!\!\} \cup \mathcal{P}, \Phi \cup \{\mathsf{ax}@t_0 \mapsto v\!\downarrow\}, t_0)$$
$$\text{with } \rho = \{t \mapsto t_0\},\ b\rho \text{ holds},\ \mathsf{ax} \in \mathcal{AX} \setminus dom(\Phi),\ \xi \vdash_{\Phi@t_0} u\!\downarrow, \text{ and } v \vdash_{\Phi@t_0} v\!\downarrow \qquad (\textsc{Out})$$

$$(\{\!\!\{\mathsf{in}(u,x)@t \text{ when } b;P, \mathsf{out}(w,v)@t' \text{ when } b';Q\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\tau} (\{\!\!\{P\rho, Q\rho'\}\!\!\} \cup \mathcal{P}, \Phi \cup \{x'@t_0 \mapsto v\!\downarrow\}, t_0)$$
$$\text{with } \rho = \{t \mapsto t_0, x \mapsto x'\},\ x' \text{ fresh renaming of } x,\ \rho' = \{t' \mapsto t_0\},\ b\rho \text{ and } b'\rho' \text{ hold},\ v \vdash_{\Phi@t_0} v\!\downarrow,\ u \vdash_{\Phi@t_0} u\!\downarrow \text{ and } w \vdash_{\Phi@t_0} u\!\downarrow \qquad (\textsc{Comm})$$

$$(\{\!\!\{\mathsf{event}\ \mathsf{Ev}(\vec{u})@t \text{ when } b;P\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\mathsf{Ev}(\vec{u}\downarrow)} (\{\!\!\{P\{t \mapsto t_0\}\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \quad \text{if } b\{t \mapsto t_0\} \text{ holds, and } \vec{u} \vdash_{\Phi@t_0} \vec{u}\!\downarrow \qquad (\textsc{Event})$$

$$(\{\!\!\{\mathsf{new}\ k;P\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\tau} (\{\!\!\{P\{k \mapsto x\}\}\!\!\} \cup \mathcal{P}, \Phi \cup \{x@t_0 \mapsto k'\}, t_0) \quad \text{with } x \in \mathcal{X} \text{ and } k' \in \mathcal{N} \text{ fresh} \qquad (\textsc{New})$$

$$(\mathcal{P}, \Phi, t_0) \xrightarrow{\tau} (\mathcal{P}, \Phi, t_0 + \delta) \quad \text{if } \delta > 0 \qquad (\textsc{Tic})$$

$$(\{\!\!\{P_0 + P_1\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\alpha} (\mathcal{P}', \Phi', t_0) \quad \text{if } \exists i \in \{0,1\}, (\{\!\!\{P_i\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\alpha} (\mathcal{P}', \Phi', t_0) \text{ and } P_i \notin \mathcal{P}' \qquad (\textsc{Choice})$$

$$(\{\!\!\{P \mid Q\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\tau} (\{\!\!\{P, Q\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \qquad (\textsc{Par})$$

$$(\{\!\!\{!\,P\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \xrightarrow{\tau} (\{\!\!\{!\,P, P\}\!\!\} \cup \mathcal{P}, \Phi, t_0) \qquad (\textsc{Repl})$$

Figure 2: Operational semantics of the applied pi-calculus with time

(2) *time progresses*: the sequence $(t_i)_{i \in \mathbb{N}}$ is not bounded.

The restriction to temporally structured traces is motivated by practice. Typically, the time progressing assumption is key when expressing fairness towards B in the motivating example ("B can get always the result by forcing the commitment"), as the property makes reference to moments arbitrarily far in the future. This also permits to consistently define the following notion of suffix:

Definition 3.8 (trace suffix). Let $T = A_0 \xrightarrow{\alpha_1} A_1 \xrightarrow{\alpha_2} \cdots$ be a (temporally structured) trace, with $A_i = (\mathcal{P}_i, \Phi_i, t_i)$. The *suffix of T at time $t \in \mathbb{R}^+$, $t \geqslant t_0$*, is the trace written $T@t$ and defined as

$$T_{padding} \cdot (A_i \xrightarrow{\alpha_{i+1}} A_{i+1} \xrightarrow{\alpha_{i+2}} \cdots)$$

where $i = \min\{j \in \mathbb{N} \mid t \leqslant t_j\}$, and $T_{padding}$ is an empty transition if $t = t_i$, and the (Tic) transition $(\mathcal{P}_i, \Phi_i, t) \longrightarrow A_i$ otherwise.

Example 3.2. We refer to the motivating example of Section 2 parametrised with $d = 1$. Consider the timed trace $T \cdot T_\infty$, where

$$T : A(x,d) \xRightarrow{\mathsf{out}(\mathsf{ax})\ \mathsf{Standby}\ \mathsf{in}(\xi)} (\{\!\!\{A'\}\!\!\}, \Phi \cup \sigma, 0.59)$$

and T_∞ is an infinite, temporally structured sequence of (Tic) transitions, $\xi \vdash_{\Phi@0.59} u$, and given fresh $y', x_r \in \mathcal{X}, r' \in \mathcal{N}$:

$$A' = \mathsf{event}\ \mathsf{Accept}(x \oplus y'); \mathsf{out}(x); \mathsf{out}(x_r); 0$$
$$\sigma = \{x_r@0.09 \mapsto r', y'@0.59 \mapsto u\}$$
$$\Phi = \{\mathsf{ax}@0.1 \mapsto \mathsf{commit}(x, r', d)\}$$

In this trace, A initiates a session with the attacker B by sampling r' at time 0.09, commits on x with r' at time 0.1, and the attacker responds with a recipe ξ at time 0.59—not long before the timeout expiring at time 0.6. However, if the timeout were d or a greater value (say, 1.1 here), the attacker could make the outcome be a chosen boolean b_0 through another trace with the actions:

- $\mathsf{out}(\mathsf{ax}_1)$ at time $t = 0.05$,
- Standby at time $t = 0.06$,
- $\mathsf{in}(\mathsf{force}(\mathsf{ax}_1) \oplus b_0)$ at time $t = 1.05$,
- $\mathsf{Accept}(b_0)$ at time $t = 2$. ◀

4 SECURITY (HYPER)PROPERTIES

4.1 A Hyperlogic

We finally present a logic for specifying hyperproperties of protocols. Its key novelty compared to similar logics is its ability to express properties about the adversary, through more developed atomic formulae and quantifiers.

Definition 4.1 (Formula). The set of *Hypertidy CTL* formulae* is given by the grammar below:

$$\begin{array}{rllll} \varphi & ::= & \forall\pi.\varphi \quad \forall x.\varphi \quad \forall X.\varphi & & \textit{quantifiers} \\ & & \varphi \mathbin{\mathsf{U}} \varphi & & \textit{until} \\ & & \varphi \wedge \varphi \quad \neg\varphi & & \textit{logical operators} \\ & & X \vdash_\pi v \quad u \vdash_\pi v \quad \alpha_\pi & & \textit{atomic formulae} \end{array}$$

with π called a *path variable*, $X \in \mathcal{X}^2$ is called a *second-order variable*, $x \in \mathcal{X}$, α is an action (as in the semantics), and u, v are terms.

The "until" operator $\varphi \mathbin{\mathsf{U}} \psi$ means that φ should hold continuously from the current time t, and for a duration δ, while ψ holds at time $t + \delta$. It is not required that φ holds at time t or $t + \delta$. *Path quantifiers* $\forall\pi$ then quantify over all traces of the studied process. The quantification is performed from the current state: for example, $\forall\pi.(\varphi \mathbin{\mathsf{U}} \exists\pi'.\psi)$ expresses that φ should hold all along any trace π, until a point where π could potentially branch to a trace π' verifying ψ. The logical operators $\wedge$ and $\neg$ have the expected semantics. The quantifiers over computations are also key features ($\forall x$ for terms, $\forall X$ for recipes). The atomic formulae of the logic are finally $\xi \vdash_\pi u$, ("the computation ξ results in u with access to the frame of π"), and α_π ("the action α occurs at the current time"). The case $\alpha = \tau$ is also used to express that no particular action occurs.

Figure 3 then formalises when a process P satisfies a Hyper*tidy* CTL* formula φ. This is done through judgements of the form $\Pi \models \varphi$ where φ is a formula and Π is a substitution from path variables to traces. We assume in particular a dedicated path variable $\varepsilon \in dom(\Pi)$ used to track the last path quantifier in scope. We also write, to restrict all traces of Π to their suffix starting at time t:

$$\Pi@t = \{\pi \mapsto \pi(\Pi)@t \mid \pi \in dom(\Pi)\}\,.$$

$$
\begin{array}{lll}
\Pi \models \forall \pi.\, \varphi & \textit{iff} & \text{writing } A \text{ the initial process of } \Pi(\varepsilon), \text{ for all traces } T \text{ of } A,\ \Pi[\pi \mapsto T, \varepsilon \mapsto T] \models \varphi \\
\Pi \models \forall x.\, \varphi & \textit{iff} & \text{for all terms } u \text{ in normal form, } \Pi \models \varphi\{x \mapsto u\} \\
\Pi \models \forall X.\, \varphi & \textit{iff} & \text{for all recipes } \xi,\ \Pi \models \varphi\{X \mapsto \xi\} \\
\Pi \models \varphi \wedge \psi & \textit{iff} & \Pi \models \varphi \text{ and } \Pi \models \psi \\
\Pi \models \neg\varphi & \textit{iff} & \Pi \not\models \varphi \\
\Pi \models \varphi \mathrel{\mathsf{U}} \psi & \textit{iff} & \text{writing } A = (\mathcal{P}, \Phi, t) \text{ the initial process of } \Pi(\varepsilon),\ \exists \delta > 0,\ \Pi@t+\delta \models \psi \text{ and } \forall t' \in (t, t+\delta),\ \Pi@t' \models \varphi \\
\Pi \models \alpha_\pi & \textit{iff} & \text{the first transition of } \Pi(\pi) \text{ is of the form } A \xrightarrow{\alpha} B \text{ for some processes } A, B \\
\Pi \models \xi \vdash_\pi u & \textit{iff} & \text{writing } A = (\mathcal{P}, \Phi, t) \text{ the initial process of } \Pi(\pi), \text{ we have } \xi \vdash_{\Phi@t} u
\end{array}
$$

Figure 3: Satisfiability relation for formulae

Definition 4.2 (satisfiability). A process P *satisfies* a formula φ, written $P \models \varphi$, when $\{\varepsilon \mapsto \varepsilon_\infty\} \models \varphi$, where ε_∞ is an infinite, temporally structured sequence of (Tic) transitions starting from P.

4.2 Useful Syntax Extensions

We now present syntax extensions encodable into our logic, that will be convenient for our case studies. We recall that, beyond classical logical operators ($\exists$, $\vee$, etc.), we already introduced in the motivating example the Progress formula, and $\mathsf{K}(x)_\pi \triangleq \exists X, X \vdash_\pi x$ expressing the non-secrecy of x at the current time in π.

Temporal Operators. The most common syntax extensions of temporal logics are the following additional temporal operators, some of them appearing in the motivating example:

$$\mathsf{F}\,\varphi = \top \mathrel{\mathsf{U}} \varphi \qquad \mathsf{G}\,\varphi = \neg(\mathsf{F}\,\neg\varphi) \qquad \varphi \mathrel{\mathsf{U}^?} \psi = (\varphi \mathrel{\mathsf{U}} \psi) \vee \mathsf{G}\,\varphi$$

$$\varphi \mathrel{\mathsf{R}} \psi = \neg(\neg\varphi \mathrel{\mathsf{U}} \neg\psi) = \psi \mathrel{\mathsf{U}^?} (\varphi \wedge \psi)$$

Intuitively, $\mathsf{F}\,\varphi$ ("finally φ") means that φ will eventually be true. Its dual operator $\mathsf{G}\,\varphi$ ("globally φ") expresses that φ is continuously true. Typically, the formula $\mathsf{F}\,\mathsf{G}\,\tau_\pi$ is satisfied by finite traces, i.e., traces as usually considered in protocol analysis that only perform a finite number of non-silent actions. The relaxed variant of "until", $\varphi \mathrel{\mathsf{U}^?} \psi$ ("φ weak until ψ"), allows φ to hold forever in case ψ never holds. The $\varphi \mathrel{\mathsf{R}} \psi$ ("φ release ψ") has a dual semantics: at time t, it states that ψ should hold at any time $t+\delta$ such that φ is continuously false in the window $(t, t+\delta)$. I.e., the first occurrence of φ "releases" the obligation of ψ to hold, hence its equivalence with $\psi \mathrel{\mathsf{U}^?} (\varphi \wedge \psi)$.

It is also possible to consider, as in real-time extensions of LTL such as (Hyper)MITL [4, 39], constrained versions of the temporal operators. The formula $\varphi \mathrel{\mathsf{U}_I} \psi$, where I is a real interval, means that φ should hold during a time lapse $\delta \in I$, and ψ holds after that. Therefore $\varphi \mathrel{\mathsf{U}} \psi$ is equivalent to $\varphi \mathrel{\mathsf{U}_{(0,+\infty)}} \psi$. The analogues $\mathsf{F}_I\,\varphi = \top \mathrel{\mathsf{U}_I} \varphi$, $\mathsf{G}_I\,\varphi = \neg\mathsf{F}_I\,\neg\varphi$ and $\varphi \mathrel{\mathsf{R}_I} \psi = \neg(\neg\varphi \mathrel{\mathsf{U}_I} \neg\psi)$ can naturally be defined. For example, at time t, the formula $\varphi \mathrel{\mathsf{R}_I} \psi$ is notably read as "*ψ should hold at any moment $t+\delta$, $\delta \in I$, except maybe when φ held at some time $t' \in (t, t+\delta)$*". Such interval constraints can most often be encoded as time conditions in processes and are therefore not necessary in our framework from an expressivity standpoint. We however describe in the technical report [10] a blockchain atomic swap protocol where this operator is convenient to express properties of the form "*the protocol is fair for all participants reacting fast enough*".

Indistinguishability. To express stronger notions of secrecy than what is possible using the K formula, we can formalise *process indistinguishability.* Typically, to model that no information can be extracted about the parameter x of a process $P(x)$, we require that the executions of $P(x_0)$ and $P(x_1)$ are indistinguishable from the adversary's view for any term x_0 and x_1. Formally, two traces are indistinguishable if, first, they perform the same interactions with the attacker (inputs and outputs):

$$\begin{aligned}\varphi_A(\pi, \pi') \ \triangleq\ \forall X. \forall Y. (&\mathsf{in}(X, Y)_\pi \Leftrightarrow \mathsf{in}(X, Y)_{\pi'}) \\ \wedge (&\mathsf{out}(X, Y)_\pi \Leftrightarrow \mathsf{out}(X, Y)_{\pi'})\end{aligned}$$

and if, second, the adversary cannot compute an equality test (started as early as possible and finished by the current time) that holds in one process and not on the other. This would indeed permit to distinguish a black-box access to one process against the other by effectively computing this test. This is expressed by the formula:

$$\varphi_E(\pi, \pi') \ \triangleq\ \forall X. \forall Y.\, X \sim_\pi Y \Leftrightarrow X \sim_{\pi'} Y$$

where $X \sim_\pi Y \triangleq \exists x, X \vdash_\pi x \wedge Y \vdash_\pi x$. We would then call *static equivalence* the conjunction of these two formulae:

$$\pi \sim \pi' \triangleq \varphi_A(\pi, \pi') \wedge \varphi_E(\pi, \pi')$$

Process indistinguishability itself may then be modelled by *trace equivalence* [23]. It states that for each trace of any of two processes P_1, P_2, there is a statically-equivalent trace in the other. We use:

$$P = (\mathsf{event\ Left}; P_1) + (\mathsf{event\ Right}; P_2)$$

to encode path quantifiers over multiple processes P_1, P_2, with $\mathsf{Left}, \mathsf{Right} \in \mathcal{F}_0$. Trace equivalence of P_1 and P_2 then rephrases as:

$$P \models \forall \pi_1. \exists \pi_2. (\mathsf{F}\,\mathsf{Left}_{\pi_1} \Leftrightarrow \mathsf{F}\,\mathsf{Right}_{\pi_2}) \wedge \mathsf{G}\,(\pi_1 \sim \pi_2)$$

This can be generalised to quantify over an arbitrary number of processes, at least regarding path quantifiers not under the scope of an "until". Under this restriction, we will thus use a syntactic sugar $\forall \pi : Q.\, \varphi$ to quantify specifically over the traces of Q. Going back to the motivating example, we then model the (strong) fairness towards A using indistinguishability, by considering:

$$P_i = \mathsf{in}(x_0); \mathsf{in}(x_1); \mathsf{in}(d); A(x_i, d) \qquad i \in \{0, 1\}, d \in \mathcal{T\!X}$$

We then require that for all accepting executions π_0 of P_0, there exists an execution π_1 of P_1 that is indistinguishable from π_0 until the event Standby occurs. This is expressed by:

$$\forall \pi_0 : P_0.\, \exists \pi_1 : P_1.\, \forall z. \mathsf{F}\ \mathsf{Accept}(z)_{\pi_0} \Rightarrow \pi_0 \sim \pi_1 \mathrel{\mathsf{U}} \mathsf{Standby}_{\pi_0}$$

4.3 Fragments of Interest

As the design of our logic has been inspired by HyperCTL* [26], we can derive several sublogics analogous to its classical fragments.

Definition 4.3. Hyper*tidy* LTL contains the formulae in *prenex form*, i.e., $Q_1\pi_1 \dots Q_n\pi_n.\varphi$, $Q_i \in \{\forall, \exists\}$, φ without path quantifiers.

It notably encompasses indistinguishability properties, but not liveness. Another important fragment is the non-relational fragment *tidy* CTL*, that is, the set of properties that do not express relations between different traces (through shared variables):

Definition 4.4. *tidy* CTL* contains the formulae φ such that for all subformulae of φ of the form $\forall\pi.\psi$, all (path, regular, second-order) variables in ψ that are not π are bound by a quantifier in ψ.

This fragment is for example sufficient to express all properties of our motivating example, but not indistinguishability, making it incomparable to Hyper*tidy* LTL. However, both subsume safety properties, also targeted by our automated proofs (Section 7):

Definition 4.5. *tidy* LTL contains the formulae of the form $\forall\pi.\varphi$ where φ does not contain path quantifications.

In addition of the examples of safety properties presented all across this paper, *tidy* LTL can also naturally express *resilience* assumptions on a channel c, as used in some analyses of fair non-repudiation protocols [7]. This intuitively means that, during a trace π, network manoeuvres are made to ensure that any message sent on c is received eventually.

5 THE MODEL AT WORK

We now formalise several protocols, featuring various flavours of reachability, liveness and indistinguishability properties, expressed uniformly in our logic. An additional blockchain atomic swap protocol, more technical, is provided in the technical report [10].

5.1 Yet Another Protocol for String Sampling

Similarly to the motivating example, we model here a sampling protocol between two mutually suspicious parties [47], based this time on *Verifiable Delay Functions* (VDF). A VDF is simply a pseudo-random function whose output cannot be computed faster than a chosen time d, *but* that can be verified efficiently (i.e., without spending time d to recompute it). In our model, it would simply be a function symbol with a non-null cost.

$$\mathcal{F} = \mathsf{VDF}[x,d], \mathsf{verify}[x,y], \mathsf{h}[x,y] \qquad \mathit{cost}(\mathsf{VDF}) = d$$
$$\mathcal{R} = \mathsf{verify}(\mathsf{VDF}(x,d),x) \to \mathsf{ok}$$

The symbol h models a binary hash function, with no associated rewrite rules, thus expressing a random-oracle assumption. Concretely, the protocol proceeds as follows. Two parties A and B send random strings r_A, r_B to each other, in plaintext. Given a time parameter d, each party only accepts the other's string for a duration of d after sending their own, and the result of the protocol is then $\mathsf{VDF}(\mathsf{h}(r_A, r_B), d)$. This way, provided A or B is honest, the result cannot be predicted before expiration of the timeout. We formalise unpredictability through the following process, encoding a security game where the adversary plays the role of B and guesses the result.

$$\begin{aligned} A(d) = {} & \mathsf{new}\ r_A;\ \mathsf{out}(r_A)@t;\ \mathsf{in}(r_B); \\ & \mathsf{in}(\mathit{guess})@t'\ \mathsf{when}\ t' < t + d; \\ & \mathsf{event}\ \mathsf{Challenge}(\mathsf{verify}(\mathit{guess}, \mathsf{h}(r_A, r_B))); 0 \end{aligned}$$

Unpredictability is then simply modelled by the following formula, to be satisfied by the process $\mathsf{in}(d); A(d)$, with $d \in \mathcal{TX}$:

$$\forall\pi.\mathsf{G}\ \neg\mathsf{Challenge}(\mathsf{ok})_\pi$$

5.2 Randomness Beacon

To sample strings publicly, one may apply an extraction function to entropy sources like stock markets [25] or blockchains [19]. These sources are believed to produce high entropy but are also manipulable to some extent. To make such manipulations virtually useless, one may use a VDF (see Section 5.1) as an extractor [17]. If its computation takes longer than the time necessary to influence the entropy source, an active attacker might indeed bias the source, but without actual insight on the impact of their action.

Concretely, consider, given an initial seed s, a *randomness beacon* [17] that publishes a new random string at regular time intervals (say $d \in \mathbb{R}^+$). It is required that the successive published strings remain unpredictable across the successive rounds, meaning:

n-round unpredictability: given the transcript of the first n sampled strings, one cannot distinguish (in time less than d) the last string from a random one.

Weaker variants may also be considered, based on computability instead of indistinguishability from random ("the adversary should not be able to guess the actual value of the n^{th} string"), similarly to how we formalise unpredictability in the string-sampling protocol of Section 5.1. The actual protocol simply proceeds iteratively by taking the last emitted string s (initially, the seed), and computes a new string r as the output of the VDF on input s with time parameter d. It then reveals r (and the corresponding proof), sets $s = r$, and repeats the process. We model unpredictability using the following two processes A_i, $i \in \{0,1\}$, given a private channel $e \in \mathcal{N}$ to carry out the state of the beacon across multiple rounds, and $d \in \mathcal{TX}$.

$$\begin{aligned} A_i &= \mathsf{in}(d); (\mathit{Init} \mid \mathit{Chal}_i(d) \mid !\, R(d)) \\ \mathit{Init} &= \mathsf{new}\ s;\ \mathsf{out}(e,s);\ \mathsf{event}\ \mathsf{Reveal};\ \mathsf{out}(s); 0 \\ R(d) &= \mathsf{in}(e, \mathit{state});\ \mathsf{let}\ r = \mathsf{VDF}(\mathit{state}, d)\ \mathsf{in} \\ &\qquad \mathsf{out}(e,r);\ \mathsf{event}\ \mathsf{Reveal}; \mathsf{out}(r); 0 \\ \mathit{Chal}_i(d) &= \mathsf{in}(e, \mathit{state}); \\ &\qquad \mathsf{new}\ r_0;\ \mathsf{let}\ r_1 = \mathsf{VDF}(\mathit{state}, d)\ \mathsf{in} \\ &\qquad \mathsf{event}\ \mathsf{Challenge}@t; \mathsf{out}(r_i); \\ &\qquad \mathsf{event}\ \mathsf{Stop}@t'\ \mathsf{when}\ t' < t + d; 0 \end{aligned}$$

The process *Init* initialises the seed, R carries out one round of the generation, and Chal_i reveals a string r_i and challenges to the adversary to guess, before the event Stop, whether it is the last pseudo-random string r_1, or a real random string r_0. The process however allow the successive random strings r to be revealed gradually, whereas the security game requires them to be released in one go, after the challenge is issued. We enforce this property through what can be seen as *trace restriction H*, similarly to the feature of verification tools such as ProVerif or Tamarin [12, 16]. Concretely, for all $i \in \{0,1\}$, the following formula should be satisfied:

$$\forall\pi : A_i.\, H_\pi \Rightarrow \exists\pi' : A_{1-i}.\, H_{\pi'} \wedge (\pi \sim \pi'\ \mathsf{U}\ \mathsf{Stop}_\pi)$$

where $H_\pi = \mathsf{F}\ \mathsf{Stop}_\pi \wedge (\neg\mathsf{Reveal}_\pi\ \mathsf{U}\ \mathsf{Challenge}_\pi)$.

5.3 Sealed-Bid Auctions

We sketch a protocol for sealed-bid auctions [18], where multiple parties engage in a Vickrey auction without the aid of a trusted auctioneer. This protocol uses the same theory for timed commitment that we used for our motivating example. We assume that the bidding phase has a (conservatively set) maximum duration of d.

(1) All participants send a d-timed commitment on their bid.
(2) After time d, honest participants open their commitment.
(3) Then, they force the commitments of parties that did not provide an opening, and compute the winner.

We give here a general model of the protocol for an unbounded number of participants. If a process P models the behaviour of an honest participant of an auction parametrised by an ending time t_f, the overall process can be modelled as follows, given $d \in \mathcal{TX}$:

$$\begin{aligned} A[P] = {} & \mathsf{in}(d);\ \mathsf{event\ Start@}t_0; \\ & (\mathsf{event\ Stop@}t_1\ \mathsf{when}\ t_1 < t_0 + d; 0)\ |\ !\,P(t_0 + d) \end{aligned}$$

The auction lasts at most a duration d (event Stop), involves an arbitrary number of honest participants (process $!\,P(t_0 + d)$) and an implicit coalition of an arbitrary number of dishonest participants (impersonated by the adversary). The process defining one participant, with bid x for an auction ending at time t_f, is:

$$\begin{aligned} Q(x, t_f) &= \mathsf{new}\ r;\ \mathsf{out}(\mathsf{commit}(x, r, d)); \\ & \qquad (\mathsf{out}(x)@t\ \mathsf{when}\ t > t_f; \mathsf{out}(r); 0\ \ |\ \ !\,R(t_f)) \\ R(t_f) &= \mathsf{in}(y)@t\ \mathsf{when}\ t < t_f;\ \mathsf{event\ Standby}(y); \\ & \qquad (\mathsf{in}(z); (\mathit{Open} + F)) + F \\ \mathit{Open} &= \mathsf{in}(s); \mathsf{if\ open}(y, z, s) = \mathsf{ok\ then\ event\ Get}(z); 0\ \mathsf{else}\ F \\ F &= \mathsf{event\ Get}(\mathsf{force}(y)); 0 \end{aligned}$$

The process R receives a single bid and opens it (event Get); the process is replicated to model an arbitrary number of receptions. The desired security properties are then that:

(1) no one can be prevented from computing the winner;
(2) the bids remain secret until the end of the auction.

To express the liveness property (1), we refer to process $A[P]$, with

$$P(t_f) = \mathsf{new}\ \mathit{bid}; Q(\mathit{bid}, t_f)$$

that should verify the formula $\forall\pi$. Progress $\Rightarrow \varphi$, with:

$$\varphi = \mathsf{G}\ (\forall x, y, d.\ \mathsf{Standby}(\mathsf{commit}(x, y, d))_\pi \Rightarrow \mathsf{F}\ \mathsf{Get}(x)_\pi)$$

Regarding Property (2), similarly as before, we model time limited strong secrecy through a security game where the adversary may spawn a number of (honest) auction participants of their choice, crafts, for each, a pair of bids x_0, x_1, and then attempts to distinguish two hypothetical auctions where the first and second bids are used, respectively. We therefore consider the following processes P_i:

$$P_i(t_f) = \mathsf{in}(x_0);\ \mathsf{in}(x_1);\ Q(x_i, t_f) \qquad\qquad i \in \{0, 1\}$$

and model the property by the following formula:

$$\forall\pi_0 : A[P_0].\ \exists\pi_1 : A[P_1].\ \mathsf{F}\ \mathsf{Stop}_{\pi_0} \Rightarrow \pi_0 \sim \pi_1\ \mathsf{U}\ \mathsf{Stop}_{\pi_0}$$

5.4 Fair Contract Signing

To conclude, we present a contract signing protocol [18], where two parties A and B want to sign a common contract Γ. Neither of the parties wants to sign the contract if it is not ensured that the other party will also sign. For that we use a *verifiable-timed signature* (VTS) [57]: this notion is similar to timed commitments, except that whatever is inside the commitment is guaranteed to be a valid signature on a chosen string. The protocol fixes a security time parameter $d = 2^\eta$ (typically $\eta = 128$), and proceeds as follows:

(1) A computes a VTS with time parameter d on Γ and sends it to B, and so does B, sending it to A;
(2) if $d = 1$, or if either party failed to send a valid VTS, force the one obtained at the previous round. Otherwise, repeat the steps with time parameter $d' = \frac{d}{2}$.

One round of the protocol is unfair: whoever signs first can be worse off, as the other may simply force the VTS and not send their own signature. This is compensated by making a force opening prohibitively hard in the first round ($d = 2^\eta$), and then incrementally easing it until reaching $d = 1$. If one party interrupts the protocol at step i, they may obtain a VTS with parameter $d = 2^{\eta-i}$, but the other party obtained a VTS with parameter $2d = 2^{\eta-i+1}$ during the previous round. In short: one may worse off a party only by a factor of 2 in terms of computation effort. To model this, we use:

$$\mathcal{F} = \mathsf{sign}[x, y, z], \mathsf{pk}[x], \mathsf{VTS}[x, d], \mathsf{verify}[x, y, z], \mathsf{force}[x]$$

$$\begin{aligned} \mathcal{R} = {} & \mathsf{force}(\mathsf{VTS}(\mathsf{sign}(x, y, z), d)) \xrightarrow{+d} \mathsf{sign}(x, y, z) \\ & \mathsf{verify}(\mathsf{VTS}(\mathsf{sign}(x, y, z), d), x, \mathsf{pk}(z)) \rightarrow \mathsf{ok} \end{aligned}$$

The second rewrite rule checks that a given a VTS contains a signature on x by the owner of the public key $\mathsf{pk}(z)$. We use three arguments for the sign function, the second one being a placeholder for randomness. We then model fairness through a security game:

(1) the adversary B engages in protocol rounds with an honest agent A (and completes at least one);
(2) when B computes a signature of A on Γ, say, at time T, they can challenge A: the latter wins the game *iff* it manages to compute B's signature on Γ in time $2T$ starting from the challenge time.

To model this practically, we let $sk_A \in \mathcal{N}$ be a name modelling the signing key of A, and $\Gamma, sk_B \in \mathcal{F}_0$ be two constant modelling the contract and the signing key of the adversary B. We use internal communications on private channels $e_1, e_2 \in \mathcal{N}$ to model state passing from one round to the other, which enforces in addition that rounds are executed sequentially. The protocol can then be modelled as follows, writing $S_{I,d} = \mathsf{VTS}(\mathsf{sign}(\Gamma, r_I, sk_I), d)$ a term modelling a signature of the agent I on Γ:

$$\begin{aligned} A &= \mathsf{out}(\mathsf{pk}(sk_A)); (\mathit{Init}\ |\ \mathit{Chal}\ |\ !\,R) \\ \mathit{Init} &= \mathsf{in}(d);\ \mathsf{out}(e_1, \tfrac{d}{2}); \\ & \qquad \mathsf{new}\ r_A;\ \mathsf{new}\ r_B;\ \mathsf{out}(S_{A,d});\ \mathsf{out}(e_2, S_{B,d}); 0 \\ R &= \mathsf{in}(e_1, d);\ \mathsf{new}\ r_A;\ \mathsf{out}(S_{A,d}); \\ & \qquad \mathsf{in}(\mathit{vts});\ \mathsf{if\ verify}(\mathit{vts}, \Gamma, \mathsf{pk}(\mathsf{sk}_B)) = \mathsf{ok\ then} \\ & \qquad\quad \mathsf{out}(e_1, \tfrac{d}{2});\ \mathsf{in}(e_2, \mathit{state});\ \mathsf{out}(e_2, \mathit{sig}); 0 \\ \mathit{Chal} &= \mathsf{in}(e_2, \mathit{state}); \mathsf{in}(\mathit{sig}); \\ & \qquad \mathsf{event\ Challenge}(\mathit{check})@T; \\ & \qquad\qquad \mathit{with\ check} = \mathsf{verify}(\mathsf{VTS}(\mathit{sig}, 0), \Gamma, \mathsf{pk}(sk_A)) \\ & \qquad \mathsf{event\ Lose}(\mathsf{force}(\mathit{state}))@t\ \mathsf{when}\ t < 3T; 0 \end{aligned}$$

The initial output of $\mathsf{pk}(sk_A)$ reveals the verification key to the adversary, and *Init* encodes a first, mandatory round of the protocol. The process $!\,R$ then models optional rounds: the agent retrieves the current value of the parameter on e_1, sends their VTS and verifies the adversary's, and then updates the internal state on e_2. At any point, the adversary may start to execute *Chal*: they provide at time T the signature *sig* of A, who is then tasked to retrieve the adversary's signature within a $2T$ window, i.e., by time $t = 3T$. Fairness is thus stated as a liveness property: "*whenever the adversary B issues a challenge, A can meet the challenge in time*".

$$\varphi = \mathsf{G}\ (\mathsf{Challenge}_\pi(\mathsf{ok}) \Rightarrow \mathsf{F}\ \exists x.\ \mathsf{Lose}_{\pi'}(x))$$

To avoid the situation of A losing by simply waiting $2T$ units of time idly, the actual security formula includes an assumption of what we call here *active-time progress*:

$$\forall \pi.\ (\mathsf{Progress} \wedge \mathsf{Active}_\pi) \Rightarrow \varphi$$

Here $\mathsf{Active}_\pi = \mathsf{G}\ (\neg \mathsf{stuck} \Rightarrow \mathsf{F}\ \neg\tau_\pi)$ (*time activeness*), namely, a non-stuck process always perform at least one action in the future.

6 DECIDABILITY AND COMPLEXITY

In this section, we study the decidability of properties expressed in our framework, i.e., the following decision problem *Verif*:

> Input: A signature $\mathcal{F}$, a rewriting system $\mathcal{R}$ and a process P built from them, a Hyper*tidy* CTL* formula φ.
> Question: $P \models \varphi$?

This can be seen as the analogue of the model checking problem of a Kripke structure against a standard HyperCTL* formula [26, 35]. This latter problem is known to be decidable in general [26], and even **PSPACE** complete in the non-relational fragments LTL and CTL*, using techniques mostly from automata theory. One may wonder to which extent these results carry out to the more involved *tidy* logics, in particular in regards of results for extensions of temporal logics with first-order quantifiers [40]. However:

Proposition 6.1. *The Verif problem is undecidable for tidy LTL (and thus tidy CTL*), even for processes without replication.*

Indeed, the undecidability of *deducibility* for convergent rewriting systems is a standard result in (untimed) protocol analysis [2]; said differently, *Verif* is undecidable even for inputs of the form:

$$P = \mathsf{out}(u_1); \ldots; \mathsf{out}(u_n); 0 \qquad \varphi = \exists \pi.\ \mathsf{F}\ (\exists X.\ X \vdash_\pi u)$$

Note that this does not rely on time constraints, hence our comparison with untimed logics such as LTL and CTL*. We propose in this section further results that draw a preliminary picture of the possibilities and limits of the problem in terms of decidability. We assume familiarity with notions of complexity theory, that are detailed in the technical report [10]. Still, as a minimal reminder, we give here common relations between complexity classes, including **PH** (polynomial hierarchy) and **EXPH**(poly) (polynomially-bounded exponential hierarchy):

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$$
$$\mathbf{EXP} \subseteq \mathbf{NEXP} \subseteq \mathbf{EXPH} \subseteq \mathbf{EXPH}(\mathrm{poly}) \subseteq \mathbf{EXPSPACE}$$

6.1 Reduction to Constraint Solving

Despite the overall undecidability, powerful techniques have been developed in protocol analysis to tackle more specific classes of security properties. A popular one for untimed bounded processes is to characterise security as a finite set of adversarial constraints [13, 22, 23]. Examples include the reduction of trace equivalence (expressible in Hyper*tidy* LTL) to the *equivalence of vectors of constraint systems* [22], a subsequent work establishing in a similar way its co**NEXP** completeness for a class of rewriting systems [23]. Another similar result establishes the co**NP** completeness of some safety properties (expressible in *tidy* LTL), by reducing them to the solvability of simpler constraints [13]. In this section, we generalise this set of techniques by exhibiting a natural notion of constraint crisply characterising the decision of arbitrary hyperproperties expressed in Hyper*tidy* CTL*. This provides in particular for the first time such reductions for, e.g., liveness properties.

Our constraints are, intuitively, first order formulae expressing time conditions and computability requirements. They are thus similar to Hyper*tidy* CTL* formulae, but evacuate all considerations related to traces (path variables, "until", actions), and keep track of relations between different attacker computations.

Definition 6.2 (Constraint). The grammar of *constraints* is:

$$\begin{array}{llllll} \Gamma & ::= & \forall X.\,\Gamma & \forall x.\,\Gamma & & \textit{term quantifiers} \\ & & \Gamma \wedge \Gamma & \neg\Gamma & & \textit{logical operators} \\ & & u \vdash_{\Phi@t} v & X \vdash_{\Phi@t} v \quad b & \xi = \zeta & \textit{atomic constraints} \end{array}$$

with u, v terms, $x \in \mathcal{X}$, $X \in \mathcal{X}^2$, $t \in \mathcal{TX}$, b time condition, ξ, ζ recipes that may contain second order variables, and Φ dated frame whose domain contains elements of the form $\mathsf{ax}@t$, where $t \in \mathcal{TX}$ instead of $\mathbb{R}^+$. We say that Γ is *valid* when it is satisfied by interpreting $\vdash_{\Phi@t}$ as dated computability, and $=$ as syntactic equality.

The classical approach to reduce security to the validity of constraints is to define a *symbolic semantics* abstracting all sources of unboundedness of the baseline semantics—here, Rules (Tic) and (In)—by constraints characterising them. We define one in the technical report [10] for our timed calculus, and formalise its soundness and completeness w.r.t. the regular semantics for bounded processes. We prove in particular the following result:

Theorem 6.3. *The Verif problem is decidable for bounded processes with an oracle for testing the validity of constraints.*

Solving the underlying constraints is in particular undecidable in general by Proposition 6.1. A promising direction for future work is therefore to study their solvability for practical, more restricted fragments to obtain decidability results similar to the aforementioned ones, for protocols built on top of timed cryptography.

6.2 Negative Results

As a complement, we now establish complexity lower bounds for *Verif*, in order to give a theoretical insight on the difficulty of various instances of the problem. We focus here on notable fragments:

(1) The *pure π-calculus* corresponds to having an empty signature (but we still allow arbitrary event symbols for convenience). Although its interest is limited in terms of security modelling, the resulting framework is closer to standard logics such as HyperCTL*, making their comparative study more insightful.

(2) *Subterm convergence* is a syntactic restriction on rewrite rules $\ell \to r$, stating that r should either be a strict subterm of ℓ, or a term without variables in normal form. All rules presented in examples in this paper are subterm convergent, and several decidability results are known in the untimed case under this assumption [2, 22, 23], making the class relevant to consider.
(3) *Bounded processes* are processes without replication [22, 23]. Replication makes even the pure π-calculus Turing complete, yielding many undecidability results. Yet, the bounded fragment is not trivially decidable (recall Proposition 6.1), and corresponds to protocols with a fixed number of participants.

In addition, all of our complexity lower bounds will be stated for *guarded formulae*, a standard syntactic criterion of formulae, required in practical tools such as Tamarin [12]. Its aim is to restricts term quantifiers $\forall x, \forall X$ to configurations of the form $\forall \vec{\omega}.\varphi_0 \Rightarrow \varphi$, where the so-called *guard* φ_0 ensures that only a finite number of instances of the variables $\vec{\omega}$ need to be considered when proving φ. We propose below a possible formalisation in our framework.

Definition 6.4 (Guarded quantifier). A Hyper*tidy* CTL* formula

$$\forall \omega_1 \dots \forall \omega_n . \varphi_0 \Rightarrow \varphi$$

is called a *guarded quantifier*, with $\vec{\omega} = \omega_1, \dots, \omega_n$ are regular or second-order variables appearing in the *guard* φ_0. We also require that φ_0 is of either of the following forms:

(1) $u \vdash_\pi x$, with u a term not containing variables of $\vec{\omega}$, and $x \in \mathcal{X}$;
(2) $\mathsf{in}(X, Y)_\pi$ or $\mathsf{out}(X, Y)_\pi$, X, Y second-order variables;
(3) $\mathsf{Ev}(\vec{x})_\pi$ for some event Ev and (regular) variables $\vec{x}$;
(4) or $\mathsf{F}\ \varphi_1$, where φ_1 of one of the above three forms.

Definition 6.5 (Guarded formula). A Hyper*tidy* CTL* formula is said to be *guarded* if all of its quantifiers over regular and second-order variables are either guarded, or of the form $\forall X. \neg X \vdash_\pi u$.

We stress in particular that $\exists \vec{\omega}. \varphi_0 \wedge \varphi = \neg(\forall \vec{\omega}. \varphi_0 \Rightarrow \neg\varphi)$ is also guarded. Similarly, $\forall \vec{\omega}, \neg\varphi_0$ may be interpreted as the guarded formula $\forall \vec{\omega}, \varphi_0 \Rightarrow \bot$. Our formalisation is purposely very restrictive, and could be easily generalised to broader uses of temporal or logical operators in guards, for example. Our goal was to strengthen our complexity lower bounds to make them relevant w.r.t. future decidability results relying on potentially more permissive definitions. All *tidy* CTL* formulae presented throughout this paper can however be interpreted as guarded, showcasing that it already captures many examples. Guarded formulae are however unable to express static equivalence (cf Section 4.2: the formula φ_E involves a quantifier alternation). We obtain the following results, proved in the technical report [10] (also valid in the untimed setting).

Proposition 6.6. *For bounded processes of the pure π-calculus and guarded formulae,* Verif *is **PSPACE** hard even in tidy LTL.*

This result may seem surprising, as several classical reachability properties such as weak secrecy are known to be co**NP** complete even with some non-trivial term algebras [54]. The **PSPACE** hardness above stems from the possibility to use quantifiers over terms in (guarded) formulae, thus easily permitting to encode the validity of *Quantified Boolean Formulae*, the prototypical **PSPACE** complete problem. However, we establish a significantly stronger lower bound when a non-trivial term algebra is additionally used:

Proposition 6.7. *For bounded processes of a term algebra with a subterm convergent rewriting system, and for guarded formulae,* Verif *is **EXPH**(poly) hard in tidy CTL* and Hypertidy LTL.*

Existing results for trace equivalence already imply the hardness of Hyper*tidy* LTL w.r.t. some levels of the exponential or polynomial hierarchies, depending on whether subterm convergent rewriting, or none, is used [24]. We therefore non-trivially extend these lower bounds. Our negative results are summarised by the following table:

rewriting	*tidy* LTL	*tidy* CTL*	Hyper*tidy* LTL/CTL*
any	undecidable		
subterm	**PSPACE** hard	**EXPH**(poly) hard	**EXPH**(poly) hard
none	**PSPACE** hard		

7 AUTOMATED VERIFICATION

We conducted a preliminary evaluation of our timed model with Tamarin, a popular verification tool for untimed protocols [12]. Our approach is based on successive counterexample-guided refinements of an untimed model. Tamarin supports the syntax of the applied π-calculus, but internally uses *multiset rewrite rules* (MSR):

$$[\ \vec{u}\ ] -[\ \vec{v}\ ]\to [\ \vec{w}\ ]$$

An execution state in Tamarin can be seen as a multiset of terms, and applying the above rule removes the terms $\vec{u}$ from it, adds in addition the terms $\vec{w}$, and labels the operation with the *facts* $\vec{v}$ (similarly to events in our model). Cryptographic primitives can be specified in a term algebra similar to the one we use. Regarding security properties, Tamarin supports among others a first-order logic with explicit timestamps (with a restriction similar to our notion of guarded formulae in Section 6.2). This allows in particular to express most properties of interest built from the "until" operator. Given a *tidy* LTL formula $\forall \pi. \varphi$ and a process P (not necessarily bounded), we used the following approach to (dis)prove $P \models \forall \pi. \varphi$.

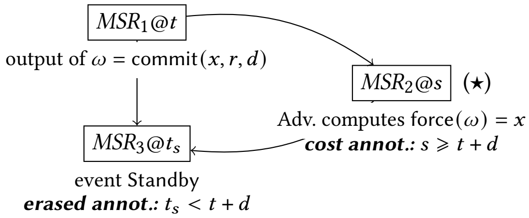
(1) When a symbol $\mathsf{f}[\vec{x}]$ has a non-null cost, we declare it as a *private symbol* in Tamarin, meaning that the adversary cannot use it to build recipes. We then add a MSR:

$$[\ \mathsf{in}(\vec{x})\ ] -[\ \mathsf{App\_f}(\vec{x})\ ]\to [\ \mathsf{out}(\mathsf{f}(\vec{x}))\ ] \qquad (\star)$$

This allows the adversary to apply f arbitrarily by using this rule as a form of oracle that outputs $\mathsf{f}(\vec{x})$ upon reception of $\vec{x}$. This way, all applications of f are materialised in traces by a fact App_f. For simplicity, we modelled all timed cryptography this way, i.e., by making without costs on rewrite rules.
(2) We then erase time annotations from P (timestamps, time conditions). We record this erasure, syntactically, by adding fresh labels on the corresponding MSR (called Timestamp facts in the following). This results in an untimed process $\bar{P}$ whose set of traces *supersedes* that of P. We then verify $\bar{P} \models \forall \pi. \varphi$ using Tamarin. Assuming the analysis terminates and conclusive:
   (a) either we get a proof;
   (b) or Tamarin returns a trace violating φ, displayed as a DAG (V, E). Its vertices are MSR instances timestamped by temporal variables, and an edge $(r_1@t_1 \to r_2@t_2) \in E$ can be seen here as a temporal dependency $t_1 < t_2$.

In case (2a), we conclude that $P \models \forall\pi.\,\varphi$. This is sound for *tidy* LTL formulae, but not in general in presence of arbitrary path quantifiers as in liveness or indistinguishability properties. This was however sufficient to cover several protocols among those presented in Section 5, provided we replace indistinguishability properties by weaker variants based on computability (K formula).

In case (2b), we verify that the attack graph is consistent with the erased time annotations (marked by Timestamp facts) and the computation costs (marked by App_f facts when performed by the adversary). This amounts to a simple SMT solving of these time constraints and of the inequalities induced by the graph edges. For example, when analysing the motivating example, Tamarin finds a potential violation of fairness towards A; that is, a way to obtain A's commitment x before the Standby event. A simplified attack graph is informally pictured below, with time annotations added:


$MSR_1@t$
output of ω = commit(x, r, d)
$MSR_2@s$
$(\star)$
Adv. computes force$(\omega) = x$
cost annot.: $s \geqslant t + d$
$MSR_3@t_s$
event Standby
erased annot.: $t_s < t + d$

If the overall induced time constraint is satisfiable by an instantiation of the timestamps as real numbers, we conclude that $P \not\models \forall\pi.\,\varphi$. However, in the case of the above attack graph, the constraint is:

$$(t < s < t_s) \wedge (s \geqslant t + d) \wedge (t_s < t + d)$$

which cannot be satisfied by any instantiation of t, t_s, s. This means that the attack graph does not correspond to a valid *timed* trace, and we generate a formula ψ that excludes this inconsistent trace T exactly. It is a straightforward Tamarin formula stating that, if all of the facts in T occur, then at least one inequality induced by the edges of the graph of T should be false. We then add ψ as a *trace restriction*, which intuitively means repeating the verification to prove $\bar{P} \models \forall\pi.\,(\psi \Rightarrow \varphi)$. The results we obtained with this method when analysing protocols of Section 5 are collected in Figure 4. They indicate how many times Tamarin was additionally run in our iterative approach to eliminate invalid attacks at Step 2b of the procedure, and how many auxiliary lemmas were added manually to guide Tamarin's solver. All modelling files can be found online:

https://gitlab.com/irakoton/20-timed-dy/-/releases/tech-report

	Protocol	Result	# add. rounds	# lemmas
fairness	string sampling	✓	1	1
	sealed-bid auction	✓	2	1
	fair contract signing	✗	–	–
unpredict.	VDF string sampling	✓	1	0
	randomness beacon			
	1-round	✓	1	0
	unbounded	?	1	–

✓ property verified ? timeout ✗ out of scope

Figure 4: Verification of *tidy* LTL properties using Tamarin

Discussion. We currently conducted the steps of our approach manually (model writing, test and exclusion of invalid attacks), although automating these steps would however only require a mostly non-technical engineering effort. The limitation to *tidy* LTL is more significant, excluding for example the contract signing protocol and its only liveness security property. An important direction for future work is hence to study the decidability of constraint solving to verify examples outside the scope of our embedding in Tamarin, by relying on our results in Section 6.1. This notably still remains a challenge even in the untimed case, although decision procedures for specific hyperproperties could be relevant starting points [23]. One could also rely on the support of Tamarin for liveness [7] and indistinguishability; this may however require to discard our counterexample-guided approach in favour of a *static* generation of trace restrictions characterising timed behaviours.

8 RELATED WORK

Timed Cryptographic Primitives. Time-lock puzzles are the timed equivalent of an encryption scheme and have been introduced in the work of Rivest et al. [53]. This primitive has recently received a lot of attention, exploring constructions from new assumptions [15] or with new structural properties, such as homomorphism [49] or identity-based key derivation [20]. The security of this primitive has been recently extended to CCA-security [41], non-malleability [36], and UC-security [14]. Timed commitment is a related primitive [18, 37], recently extended to the setting of verifiable timed signatures [57]. Another related notion is the recently introduced verifiable delay functions [17], that cannot encrypt messages but allow for efficient verification of the computation.

Timed Symbolic Models. There exist several timed models to reason about protocols [5, 11, 30]. However, these models aim to show the physical proximity between participants (*distance bounding*), by reasoning about network delays. These models do not consider timed cryptographic primitives, and are not supported by hyperlogics. One may still mention [51], that proves properties for distance-bounding protocols in Tamarin, recalling our implementation in Section 7. However, in their (simpler) setting not involving timed cryptography, they can characterise time conditions statically, without the need to go over a refinement loop as us. An example closer to ours may be [48], proposing a modelling of timed protocols, based on MSR like Tamarin. The security properties that it can express are however too limited (a finite set of trace properties).

Fairness in Symbolic Models. Fairness has been intensively studied in the context of security protocols [21, 38, 45, 46, 56]. These works often characterise it similarly as us, i.e., as a formula in a temporal logics similar to CTL*. They however build on finite models, which can mean considering only one or two protocol sessions, and/or approximating the power of the adversary by bounding the size of messages, thus often missing attacks. Some more recent work [6, 7] lifts these restrictions by formalising fairness in a variant of the applied π-calculus, which is closer to our contributions. They are however limited to properties written $\forall\pi$, Progress $\Rightarrow \varphi$ in our logic, and to protocols involving a TTP (i.e., no timed cryptography). Such applications can naturally be expressed in our framework.

Hyperproperties. Hyperproperties [27] is a unified framework encompassing classic trace properties, such as safety and liveness, relational properties, including equivalence relations commonly used to model security, and k-properties, which relate k traces of a system. Reasoning about hyperproperties therefore requires rich logics that are able to reason about multiple executions. Such *hyperlogics* include HyperCTL* [26] or its real-time analogue HyperMITL [39], but also many others [29]. Although our logic is inspired from them, there are major differences. First, we have an explicit model of cryptographic primitives, that can be referenced (through terms) in formulae. Second, we express properties about processes executed in an adversarial environment, and formulas can quantify over arbitrary adversarial computations. In particular, deciding whether an atomic formula holds is undecidable in our context without additional assumption—whereas it is a simple boolean test in usual hyperlogics—highlighting a gap in complexity. These logics are supported by a strong theoretical understanding of the limits of the problem in terms of decidability [4, 34, 35, 39, 50]. This includes among others results of decidability for relational properties, including fragments of HyperLTL, despite not carrying in general to *tidy* variants as shown in Section 6. There also exist formalisations of indistinguishability in concurrent process algebras with time, a few examples including [8, 52, 55]. They however only support a fixed set of cryptographic primitives, or not at all (pure π-calculus), and cannot express more refined variants of indistinguishability or hyperproperties in general.

9 CONCLUDING REMARKS

We proposed the first symbolic approach to reason about multiparty computations based on timed cryptographic primitives. We have illustrated its benefits by mechanising several protocols of interest, and by reducing decidability to a notion of constraint solving. As discussed in the corresponding sections, interesting directions for future work include investigating a static characterisation of time conditions as untimed formulas, and studying the notions of constraint solving for deciding hyperproperties.

Another exciting direction is to establish computational soundness, i.e., showing that protocols that can be proved secure in our model are also secure in the computational model. Such results have been popularised in the untimed setting [3, 44], and would be relevant to study w.r.t. standard notions of timed security [42]. A different line of work is to investigate to which extent our model may capture primitives enforcing time/memory tradeoffs [32], or to express consensus based on Nakamoto-style proofs of work.

REFERENCES

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2017. The applied pi calculus: Mobile values, new names, and secure communication. *Journal of the ACM (JACM)* 65, 1 (2017), 1–41.

[2] Martín Abadi and Véronique Cortier. 2006. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science* 367, 1-2 (2006), 2–32.

[3] Martın Abadi and Phillip Rogaway. 2000. Reconciling two views of cryptography. In *IFIP International Conference on Theoretical Computer Science*. Springer, Sendai, Japan, 3–22.

[4] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. 1996. The benefits of relaxing punctuality. *Journal of the ACM (JACM)* 43, 1 (1996), 116–146.

[5] Damián Aparicio-Sánchez, Santiago Escobar, Catherine Meadows, José Meseguer, and Julia Sapiña. 2020. Protocol Analysis with Time. In *International Conference on Cryptology in India (INDOCRYPT)*. Springer, India, 128–150.

[6] Alessandro Armando, Roberto Carbone, and Luca Compagna. 2009. LTL model checking for security protocols. *Journal of Applied Non-Classical Logics* 19, 4 (2009), 403–429.

[7] Michael Backes, Jannik Dreier, Steve Kremer, and Robert Künnemann. 2017. A novel approach for reasoning about liveness in cryptographic protocols and its application to fair exchange. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Paris, France, 76–91.

[8] Kamal Barakat, Stefan Kowalewski, and Thomas Noll. 2012. A native approach to modeling timed behavior in the pi-calculus. In *International Symposium on Theoretical Aspects of Software Engineering*. IEEE, Beijing, China, 253–256.

[9] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. IEEE Symposium on Security and Privacy (S&P).

[10] Gilles Barthe, Ugo Dal Lago, Giulio Malavolta, and Itsaka Rakotonirina. 2022. *Tidy: Symbolic Verification of Timed Cryptographic Protocols (technical report)*. Technical Report. MPI-SP, University of Bologna, IMDEA Software Institute. Available at: https://gitlab.com/irakoton/20-timed-dy/-/releases/tech-report.

[11] David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. 2011. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)* 14, 2 (2011), 1–28.

[12] David Basin, Cas Cremers, Jannik Dreier, Simon Meier, Ralf Sasse, and Benedikt Schmidt. 2019. *Tamarin prover manual.* https://tamarin-prover.github.io/.

[13] Mathieu Baudet. 2007. *Sécurité des protocoles cryptographiques: aspects logiques et calculatoires.* Ph. D. Dissertation. École normale supérieure de Cachan-ENS Cachan. In French.

[14] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. 2021. TARDIS: A Foundation of Time-Lock Puzzles in UC. In *Advances in Cryptology (EUROCRYPT)*. Springer, Zagreb, Croatia, 429–459.

[15] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. 2016. Time-Lock Puzzles from Randomized Encodings. In *ACM Conference on Innovations in Theoretical Computer Science (ITCS)*. Association for Computing Machinery, New York, USA, 345–356.

[16] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2020. *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.* https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[17] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable delay functions. In *Annual international cryptology conference (CRYPTO)*. Springer, Santa Barbara, CA, USA, 757–788.

[18] Dan Boneh and Moni Naor. 2000. Timed commitments. In *Annual international cryptology conference (CRYPTO)*. Springer, Santa Barbara, CA, USA, 236–254.

[19] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. 2015. On Bitcoin as a public randomness source. IACR Cryptol. ePrint Arch. Available at: https://eprint.iacr.org/2015/1015.pdf.

[20] Jeffrey Burdges and Luca De Feo. 2021. Delay Encryption. In *Advances in Cryptology (EUROCRYPT)*. Springer, Zagreb, Croatia, 429–459.

[21] Rohit Chadha, Steve Kremer, and Andre Scedrov. 2006. Formal analysis of multiparty contract signing. *Journal of Automated Reasoning* 36, 1 (2006), 39–83.

[22] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. 2013. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science* 492 (2013), 1–39.

[23] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2018. DEEPSEC: Deciding equivalence properties in security protocols theory and practice. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, San Fransisco, USA, 529–546. Technical Report available at https://hal.inria.fr/hal-01698177/document.

[24] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2020. The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols. In *Logic, Language, and Security*. Springer, France, 127–145.

[25] Jeremy Clark and Urs Hengartner. 2010. On the Use of Financial Data as a Random Beacon. Usenix EVT/WOTE.

[26] Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. 2014. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust (POST)*. Springer, Grenoble, France, 265–284.

[27] Michael R Clarkson and Fred B Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[28] Richard Cleve. 1986. Limits on the security of coin flips when half the processors are faulty. In *ACM symposium on Theory of computing (STOC)*. Association for Computing Machinery, USA, 364–369.

[29] Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. 2019. The hierarchy of hyperlogics. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, Vancouver, Canada, 1–13.

[30] Alexandre Debant and Stéphanie Delaune. 2019. Symbolic verification of distance bounding protocols. International Conference on Principles of Security and Trust (POST).

[31] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[32] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of space. In *Annual Cryptology Conference (CRYPTO)*. Springer, Santa Barbara, CA, USA, 585–605.

[33] Shimon Even and Yacov Yacobi. 1980. *Relations among public key signature systems*. Technical Report. Computer Science Department, Technion.

[34] Bernd Finkbeiner and Christopher Hahn. 2016. Deciding hyperproperties. arXiv preprint arXiv:1606.07047.

[35] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. 2015. Algorithms for Model Checking HyperLTL and HyperCTL*. In *Computer Aided Verification (CAV)*. Springer, San Fransisco, CA, USA, 30–48.

[36] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. 2021. Non-malleable Time-Lock Puzzles and Applications. In *Theory of Cryptography Conference (TCC)*. Springer, Raleigh, United States, 447–479.

[37] Juan A Garay and Markus Jakobsson. 2002. Timed release of standard digital signatures. In *International Conference on Financial Cryptography (FC)*. Springer, Southampton, Bermuda, 168–182.

[38] Sigrid Gürgens and Carsten Rudolph. 2005. Security analysis of efficient (Un-) fair non-repudiation protocols. *Formal Aspects of Computing* 17, 3 (2005), 260–276.

[39] Hsi-Ming Ho, Ruoyu Zhou, and Timothy M Jones. 2021. Timed hyperproperties. *Information and Computation* 280 (2021), 104639.

[40] Ian Hodkinson, Roman Kontchakov, Agi Kurucz, Frank Wolter, and Michael Zakharyaschev. 2003. On the computational complexity of decidable fragments of first-order linear temporal logics. In *International Symposium on Temporal Representation and Reasoning, and Fourth International Conference on Temporal Logic (TIME-ICTL)*. IEEE, Cairns, QLD, Australia, 91–98.

[41] Jonathan Katz, Julian Loss, and Jiayu Xu. 2020. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography Conference (TCC)*. Springer, Durham, NC, USA, 390–413.

[42] Jonathan Katz, Julian Loss, and Jiayu Xu. 2020. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography Conference (TCC)*. Springer, USA, 390–413.

[43] Steve Kremer and Robert Künnemann. 2016. Automated analysis of security protocols with global state. *Journal of Computer Security* 24, 5 (2016), 583–616.

[44] Steve Kremer and Laurent Mazaré. 2007. Adaptive soundness of static equivalence. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, Dresden, Germany, 610–625.

[45] Steve Kremer and Jean-François Raskin. 2001. A game-based verification of non-repudiation and fair exchange protocols. In *International Conference on Concurrency Theory (CONCUR)*. Springer, Aalborg, Denmark, 551–565.

[46] Steve Kremer and Jean-François Raskin. 2002. Game analysis of abuse-free contract signing. In *IEEE Computer Security Foundations Workshop (CSFW)*. IEEE, Nova Scotia, Canada, 206–220.

[47] Arjen K Lenstra and Benjamin Wesolowski. 2015. A random zoo: sloth, unicorn, and trx. IACR Cryptol. ePrint Arch..

[48] Li Li, Jun Sun, Yang Liu, and Jin Song Dong. 2015. Verifying parameterized timed security protocols. In *International Symposium on Formal Methods (FM)*. Springer, Oslo, Norway, 342–359.

[49] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. 2019. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference (CRYPTO)*. Springer, Santa Barbara, USA, 620–649.

[50] Corto Mascle and Martin Zimmermann. 2019. The keys to decidable hyperltl satisfiability: Small models or very simple formulas. arXiv preprint arXiv:1907.05070.

[51] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. 2018. Distance-bounding protocols: Verification without time and location. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, San Fransisco, USA, 549–566.

[52] Vivek Nigam, Carolyn Talcott, and Abraao Aires Urquiza. 2018. Symbolic timed observational equivalence. arXiv preprint arXiv:1801.04066.

[53] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. Massachusetts Institute of Technology. Laboratory for Computer Science.

[54] Michaël Rusinowitch and Mathieu Turuani. 2003. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science* 299, 1-3 (2003), 451–475.

[55] Neda Saeedloei and Gopal Gupta. 2013. Timed pi-Calculus. In *International Symposium on Trustworthy Global Computing (TGC)*. Springer, Buenos Aires, Argentina, 119–135.

[56] Vitaly Shmatikov and John C Mitchell. 2002. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science* 283, 2 (2002), 419–450.

[57] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. 2020. Verifiable Timed Signatures Made Practical. ACM Conference on Computer and Communications Security (CCS).