CryptoVampire: Automated Reasoning for the Complete Symbolic Attacker Cryptographic Model

Simon Jeanteur*, Laura Kovács*, Matteo Maffei*† and Michael Rawson*
*TU Wien, {firstname}.{lastname}@tuwien.ac.at
†Christian Doppler Laboratory Blockchain Technologies for the Internet of Things

***Abstract*—Cryptographic protocols are hard to design and prove correct, as witnessed by the ever-growing list of attacks even on protocol standards. *Symbolic models of cryptography* enable automated formal security proofs of such protocols against an idealized cryptographic model, which abstracts away from the algebraic properties of cryptographic schemes and thus misses attacks. *Computational models of cryptography* yield rigorous guarantees but support at present only interactive proofs and/or restricted classes of protocols (e.g., stateless ones). A promising approach is given by the *computationally complete symbolic attacker (CCSA)* model, formalized in the BC Logic, which aims at bridging and getting the best of the two worlds, obtaining cryptographic guarantees by symbolic protocol analysis. The BC Logic is supported by a recently developed interactive theorem prover, namely Squirrel, which enables machine-checked interactive security proofs, as opposed to automated ones, thus requiring expert knowledge both in the cryptographic space as well as on the reasoning side.**

In this paper, we introduce the CryptoVampire cryptographic protocol verifier, which for the first time fully automates proofs of trace properties in the BC Logic. The key technical contribution is a first-order formalization of protocol properties with tailored handling of subterm relations. As such, we overcome the burden of interactive proving in higher-order logic and automatically establish soundness of cryptographic protocols using only first-order reasoning. Our first-order encoding of cryptographic protocols is challenging for various reasons. On the theoretical side, we restrict full first-order logic with cryptographic axioms to ensure that, by losing the expressivity of the higher-order BC Logic, we do not lose soundness of cryptographic protocols in our first-order encoding. On the practical side, CryptoVampire integrates dedicated proof techniques using first-order saturation algorithms and heuristics, which all together enable leveraging the state-of-the-art Vampire first-order automated theorem prover as the underlying proving engine of CryptoVampire. Our experimental results showcase the effectiveness of CryptoVampire as a standalone verifier as well as in terms of automation support for Squirrel.

***Index Terms*—Security Protocols, Formal Methods, Computational Security, Automated Theorem Proving**

This is the technical report.

1. Introduction

Cryptographic protocols are the software interfaces used by the components of our digital world to communicate securely with one another. Unfortunately, designing such protocols is notoriously difficult and error-prone. From the classic attack on the Needham-Schroeder protocol [1], over to the ubiquitous but repeatedly broken TLS protocol [2], up to new and subtle blockchain protocols [3], the list of attacks on popular standards is ever-growing.

Formal methods have proved to be a very successful tool to guarantee properties of protocols and recently accompanied the design of protocol standards like TLS 1.3 [4], WireGuard [5], or 5G-AKA [6].

1.1. Related Work

Security properties for cryptographic protocols are typically formalized in terms of trace properties [7] or observational equivalence relations [8]. The former express guarantees on a single execution trace (possibly reflecting multiple protocol sessions) and cover secrecy of random data, authentication, and more general constraints on the order of security-relevant events. The latter express the inability of an attacker to tell the difference between two protocol configurations, such as the inability to understand which among two low-entropy secrets is used.

Formal verification of cryptographic protocols is further split into techniques working in the *symbolic* or *computational* model. Symbolic techniques typically abstract away from the algebraic properties of cryptography by reasoning over the so-called "Dolev-Yao" attacker [9], which has infinite computational resources at its disposal but is restricted to building only symbolic terms based on those already in their knowledge. This results in easier proof techniques, which enabled the design of several successful automated protocol verifiers like ProVerif [10] and Tamarin [11]. However, such a symbolic setting does not provide any computational guarantees: for instance, symbolically-secure protocols may, in fact, be attacked by leveraging weaknesses in the underlying cryptographic realization [12].

The computational cryptographic model is instead based on computational and probabilistic notions, which faithfully describe real-world implementations. In a computational setting, the attacker is a probabilistic polynomial-time Turing machine, and cryptographic proofs show that if the attacker is able to break the security of a protocol, then it can also with high probability break commonly accepted mathematical assumptions. Such a proof technique over the computational protocol model gives strong security guarantees against real-world cryptographic implementations, but it is much harder to formalize and automate.

Nonetheless, a few approaches automating computational proofs emerged over the years, such as EASYCRYPT [13], a cryptographic proof checker that simplifies reasoning about probabilistic computations in an adversarial setting and is typically used to prove the security of cryptographic schemes, and CRYPTOVERIF [14], a protocol verifier that automates the type of game-based proofs conducted by cryptographers and scales to cryptographic protocols, although it is restricted to stateless ones.

Recently, various approaches tried to merge the symbolic and computational protocol models to obtain the best of the two worlds, i.e., the ease of proofs and expressiveness of the former with the strong cryptographic guarantees of the latter. In particular, Bana and Comon-Lundh designed the *Computationally Complete Symbolic Attacker* (CCSA) model and its accompanying BC Logic [15], [16], [17], which later evolved into a fragment of higher-order logic [18]. BC Logic supports both trace properties and observational equivalence and aims at opening the door to formal, computationally sound verification, although proofs in BC Logic had long to be done by hand.

Recently, a breakthrough in the mechanization of proofs in the BC Logic has materialized in the interactive SQUIRREL proof assistant [19], which supports both trace properties and observational equivalence. SQUIRREL pushed further developments of the CCSA, such as the support for stateful protocols [20], post-quantum security [21], and proof composition [22]. SQUIRREL is an interactive theorem prover that is effective in yielding machine-checked proofs but does not provide full automation and requires expertise in both cryptography and logic.

The key technical challenge towards proof automation in the BC Logic is the expressiveness of the underlying higher-order logical fragment, which blends together a very relaxed equality theory with strict symbolic reasoning in the form of a complex subterm relation. Such a logical combination extends the full first-order theory of term algebras (including algebraic data types, such as lists) with the subterm predicate, which is inherently undecidable [23]. While proving security properties in first-order logic instead of BC Logic would come with the benefits of semi-decidability and (potential) full automation, the use of subterm relations (and other higher-order constructs) from BC Logic imposes an open challenge to automated theorem proving [24], [25], [26] as such relations are not first-order axiomatizable.

1.2. Our contributions

We introduce CRYPTOVAMPIRE, *the first automated verification tool for the BC Logic* (Section 4.2). Our work so far focuses on trace properties, leaving observational equivalence relations for future work. The design of CRYPTOVAMPIRE blends together four core contributions:

(A) We provide an encoding of the BC Logic into *first-order logic*. Our first-order encoding of BC Logic resolves the challenge of effective subterm reasoning (Section 4.1). In particular, given that subterm relations are not finitely axiomatizable in first-order logic, we provide a tailored handling of subterm relations to ensure that subterm reasoning within CRYPTOVAMPIRE can be directly supported within saturation-based first-order theorem proving [25].

(B) When moving from the higher-order setting of BC Logic to our first-order formalization, we ensure that our loss in logical expressivity when compared to the BC Logic does not impact the soundness of our proofs (Section 4.2). Namely, we show that if a protocol is proven secure in our first-order logic, then it is secure in BC Logic too (Property 5).

(C) We enhance the performance of CRYPTOVAMPIRE by introducing *dedicated reasoning procedures* within saturation. In particular, we control proof search by specialized term orderings, preprocessing techniques, and heuristics (Section 5).

(D) We conduct an *experimental evaluation* of CRYPTOVAMPIRE on all trace-based queries from the SQUIRREL library (Section 6): CRYPTOVAMPIRE can verify all these protocols in a few milliseconds. We further demonstrate how CRYPTOVAMPIRE exhibits better performance than CRYPTOVERIF on these protocols. Finally, we demonstrate the usefulness of CRYPTOVAMPIRE as automation support for SQUIRREL by proving a set of lemmas used in SQUIRREL proofs, some of which are used in the proofs of observational equivalence properties. Overall, our experimental results demonstrate that CRYPTOVAMPIRE is not only effective as a standalone cryptographic protocol verifier but can also be used to partially automate SQUIRREL proofs.

2. Overview

This section gives a high-level overview of CRYPTOVAMPIRE, while the following sections will go into the more formal details. The tool takes as input **(i)** a protocol specification composed of function declarations, capturing the cryptographic messages used in the protocol, **(ii)** constraints on those functions, expressing their cryptographic semantics, and **(iii)** a protocol description in the form of partially preordered *steps*. The given protocol description is translated into a first-order logic (FOL) formula over term algebras with tailored term evaluations, and the final proof is off-loaded to the VAMPIRE first-order theorem prover [25]. In other words, CRYPTOVAMPIRE transforms the problem of protocol verification into validity checking of term algebra properties, integrating first-order axiomatic reasoning completely within saturation theorem proving. As such, CRYPTOVAMPIRE provides fully-automated reasoning to prove the security of CCSA models, leveraging and extending subterm reasoning in term algebras within VAMPIRE.

Example 1 (Basic Hash)**.** *Let us illustrate our approach through the following simple protocol inspired by the RFID protocols described in [27]:*

$$T_j \to R : \quad \langle \mathsf{n}, \mathcal{H}(\mathsf{n}, \mathsf{k}_j)\rangle \tag{1}$$

Here, a tag T_j outputs a fresh *nonce* n *both in plain text and hashed ($\mathcal{H}$) with the key k_j shared with a reader R. For simplicity, we model a single reader R operating over multiple tags T (parameterized by j).*

Intuitively, this protocol guarantees a property called non-injective agreement [7], i.e., when R authenticates tag T_j, then T_j started an authentication session with R.

We will now illustrate the main steps of the analysis of Example 1 in CryptoVampire.

2.1. First-Order Formalization of the Protocol

Following the CCSA model, cryptography is represented *symbolically*, i.e., using a term algebra, which is built on a set of honest functions $\mathcal{F}$ and a set of nonce names $\mathcal{N}$.

2.1.1. Honest functions. The functions in $\mathcal{F}$ are *uninterpreted functions* in CryptoVampire and they represent deterministic polynomial time Turing machines.

Example 2 (Functions)**.** *To encode Example 1, we will use the following functions in $\mathcal{F}$: $\langle \_, \_\rangle, \pi_1(\_), \pi_2(\_)$ respectively denoting a pair constructor $\langle \_, \_\rangle$ and projections π_i;* ok, ko *for success and failure of tag authentication; $\mathcal{H}(\_, \_)$,* verify$(_, _, _)$ *for hash computation and verification; and* If _ Then _ Else _ *for conditionals.*

2.1.2. Honest randomness. The only source of honest randomness in CryptoVampire are *nonces*. They represent pairwise independent random bitstrings of length η, the security parameter. In the logic, nonces are constants, which are used to express certain cryptographic properties (e.g., no collision). Nonces are indexed in order to support freshness across an unbounded number of sessions and participants.

Example 3 (Nonces)**.** *In Example 1, $\mathcal{N} = \{\mathsf{n}[\_, \_], \mathsf{k}[\_]\}$. Furthermore, $\mathsf{n}[i, j]$ is indexed by the session i and the identifier j of the participating tag, and $\mathsf{k}[j]$ is only indexed by the tag j. We use nonces to represent keys as they are a source of honest randomness.*

2.1.3. Cryptographic properties. The semantics of constants and functions are expressed through FOL rules, which are typically used to express cryptographic properties.

Example 4 (Cryptographic Properties)**.** *Intuitively, in the first-order formalization of Example 1, we will encode the semantics of projection operators as $|\pi_i(\langle x_1, x_2\rangle)| = |x_i|$, for $i = 1, 2$. We write $|t|$ to denote the evaluation of t (e.g., applying a projection operator). We also formalize message authentication code through the verification condition $|\mathsf{verify}(\sigma, m, k)| \Leftrightarrow (|\sigma| = |\mathcal{H}(m, k)|)$. We capture existential unforgeability via formula (*EUF-CMA*), which we axiomatize in FOL as shown in Property 1.*

Property 1 (EUF-CMA)**.** *If $\mathcal{H}(\_, \_)$ and* verify$(_, _, _)$ *form a* MAC *scheme that is existentially unforgeable under chosen message attacks, then, for all nonce* k*, the protocol $\mathcal{P}$ satisfies the following:*

$$|\mathsf{verify}(\sigma, m, \mathsf{k})| \Rightarrow \begin{pmatrix} \mathsf{k} \sqsubseteq_{\mathsf{verify}(\_,\_,\bullet), \mathcal{H}(\_,\bullet)} m, \sigma \\ \quad \vee \exists u. (\mathcal{H}(u, \mathsf{k}) \sqsubseteq m, \sigma \wedge |u| = |m|) \end{pmatrix} \tag{2}$$

This means that, assuming $\mathcal{H}(\_, \_)$ and verify$(_, _, _)$ are EUF-CMA-secure, for any execution of $\mathcal{P}$, any message m, MAC σ, and key[1] k, if (m, σ) form a valid MAC pair with key k, then the key *symbolically appears* in a non-key position (i.e., in one of the first two arguments of verify or in the first argument of $\mathcal{H}$) in m, σ, or $\mathcal{P}$ (left disjunction operand $\sqsubseteq$), or we can find another message u that evaluates to the same bitstring as m and whose MAC with key k symbolically appears in either m or σ (right disjunction operand $\sqsubseteq_{\mathsf{verify}(\_,\_,\bullet), \mathcal{H}(\_,\bullet)}$). In other words, if verification succeeds, then either the key has been misused or the message has indeed been signed before. The cryptographic soundness of EUF-CMA is proven in Appendix F . Note that, in the CCSA model, the attacker can take any action, unless specific rules restrict its capabilities (e.g., EUF-CMA and nonce guessing).

2.1.4. Protocol steps. Within CryptoVampire, a protocol is a set of atomic *steps*, and each execution of this protocol is a valid sequence of steps. A step takes an *input* and computes an *output* (or *message*) and assignments to some *memory cells*. A step may be guarded by a *condition* (also computed from the input), which specifies under which assumptions the step is executed.

Example 5 (Steps)**.** *We can express Example 1 in terms of the following steps:*

$\underline{\mathsf{T}[i, j]}$: *($i^{th}$ execution of the j^{th} tag)*
***Condition**:* `true`
***Message**:* $\langle \mathsf{n}[i, j], \mathcal{H}(\mathsf{n}[i, j], \mathsf{k}[j])\rangle$

$\underline{\mathsf{Rs}[i, j]}$: *(Successful authentication of the j^{th} tag on the i^{th} execution)*
***Condition**:* $\mathsf{verify}(\pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j])$
***Message**:* ok

where in *stands for* input$(\mathsf{Rs}[i, j])$.

$\underline{\mathsf{Rf}[i]}$: *(No authentication on the i^{th} execution)*
***Condition**:* $\bar{\neg}\bar{\exists} j.\, \mathsf{verify}(\pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j])$
***Message**:* ko

where in *stands for* input$(\mathsf{Rf}[i])$.

Protocol steps are executed in an order captured by the relation $<$. In addition, we introduce the mutually exclusive relation $\diamond$ to relate steps, out of which at most one can be executed in the same protocol run.

Example 6 (Mutual Execution)**.** *In Example 1, the reader can only execute one branch. Thus, in each session i, the*

[1] Remember that keys are represented as nonces

authentication either succeeds or fails, and at most one tag can be authenticated, which is formalized as follows:

$$\text{for all } i,\ j \neq k,\ (\mathsf{Rf}[i] \diamond \mathsf{Rs}[i,j]) \text{ and } (\mathsf{Rs}[i,j] \diamond \mathsf{Rs}[i,k]) \tag{3}$$

There are no other ordering constraints for this protocol.

2.1.5. Protocol query. Finally, we need to provide a query, which can be any valid first-order formula. CRYPTOVAMPIRE tries to prove that this query holds for any possible execution of the protocol in any model that verifies the constraints defined in Section 2.1.3.

Example 7 (Query)**.** *For Example 1, we are concerned with non-injective agreement [7], which is formalized as:*

$$\begin{aligned}&\forall i,j.\,\mathrm{happens}(\mathsf{Rs}[i,j]) \wedge |\mathrm{cond}(\mathsf{Rs}[i,j])| \Rightarrow\\ &\exists k.\begin{pmatrix}\mathsf{T}[k,j] < \mathsf{Rs}[i,j]\\ \quad\wedge |\pi_1(\mathrm{input}(\mathsf{Rs}[i,j]))| = |\pi_1(\mathrm{msg}(\mathsf{T}[k,j]))|\\ \quad\wedge |\pi_2(\mathrm{input}(\mathsf{Rs}[i,j]))| = |\pi_2(\mathrm{msg}(\mathsf{T}[k,j]))|\end{pmatrix}\end{aligned} \tag{4}$$

Query (4) states that, if $\mathsf{Rs}[i,j]$ is selected for execution and its condition holds (i.e., authentication succeeds), then $\mathsf{T}[k,j]$ was executed before and had an output that matched $\mathsf{Rs}[i,j]$'s input (i.e., a matching authentication request was issued before).

2.2. Automated Verification of Protocol Queries

While one might expect that a first-order protocol formalization directly yields an automated verification procedure by encoding the model in a first-order theorem prover, such an encoding presents a number of technical challenges encompassing both definitions and resolution algorithms, as we review in this section.

2.2.1. Subterm reasoning. Property 1 illustrates how reasoning works within the CCSA model. We take a computational property (here $|\mathsf{verify}(\sigma,m,\mathsf{k})|$) and turn it into a symbolic analysis. This symbolic analysis often relies on variations over a *subterm relation* ($\sqsubseteq$ and $\sqsubseteq_{\mathsf{verify}(\_,\_,\bullet),\mathcal{H}(\_,\bullet)}$ in Equation 2).

To automate such rules, our first-order formalization extends the first-order theory of finite term algebras [28] with dedicated subterm reasoning. As it is not finitely axiomatizable [23], we took inspiration from [19] for some overapproximations of $\sqsubseteq$ that we then axiomatized in FOL. Moreover, we modified existing subterm reasoning in VAMPIRE [28] to allow for the flexibility required by our overapproximations, such as supporting not only $\sqsubseteq$ but also $\sqsubseteq_{\mathsf{verify}\,\_\_\bullet,\mathcal{H}\,\_\bullet}$ (cf. Section 5).

Yet, the subterm relation used for Property 1 interacts poorly with the expected equational theory of $|\_| = |\_|$ (i.e., equality over evaluated terms, later denoted as $\doteq$), as highlighted in [29] and shown next.

Example 8. *Assume $\langle\_,\_\rangle$, $\pi_1(\_)$ are respectively the tuple constructor and its first projection, and m_1 and m_2 are two* distinct *constants such that $m_1 \not\sqsubseteq m_2$. We have $m_1 \sqsubseteq \pi_1(\langle m_2, m_1\rangle)$, but also $m_2 \doteq \pi_1(\langle m_2, m_1\rangle)$. Thus, reasoning modulo $\doteq$ (i.e., equality over evaluated terms), yields $m_1 \sqsubseteq m_2$, a contradiction.*

To overcome the reasoning difficulties arising from combining subterm relations and the equality $\doteq$, we split the reasoning on terms between

- their *symbolic forms* (described in Section 3.2), on which we can *apply the subterm relation*,
- and their *evaluated form* (described in Section 4.2), denoted by $|\_|$, on which we *reason modulo $\doteq$ and do not apply subterm reasoning*.

As a result, the contradiction presented in Example 8 becomes impossible since the superposition step (substituting a with b when $a \doteq b$) is no longer allowed.

2.2.2. Soundness challenges. It has long been known that perfect notions of security are non-realistic for real-world applications [30]. Therefore, cryptographic properties are often probabilistic. The BC Logic is no exception, and its semantics are grounded in probability. Consequently, the BC Logic lies beyond classical FOL [15]. This is a critical blow, as most automated provers address classical FOL [25], [24], [26].

In Section 4.2 we propose a classical first-order encoding that overapproximates the BC Logic, then in Section 4.3 we introduce a method to regain the probabilistic cryptographic semantics while retaining the ability to reason with FOL.

2.2.3. Further optimizations. We propose reasoning heuristics (Section 5.3) within CRYPTOVAMPIRE, which allow us to eliminate most of the symbolic reasoning, leaving only what can be encoded in standard FOL (i.e., modulo $\doteq$) while remaining consistent before off-loading to the theorem prover. For that, we look for terms that might be relevant concerning the cryptographic axioms and preprocess them.

Example 9 (Preprocessing)**.** *In Example 1 we note that the term* $\mathsf{verify}(\pi_2(\mathrm{input}(\mathsf{Rs}[i,j])), \pi_1(\mathrm{input}(\mathsf{Rs}[i,j])), \mathsf{k}[j])$ *appears in the protocol specification (including the assertions and the query). Since this term has the form of the premise of axiom* (2) *of Property 1, we preprocess it in* CRYPTOVAMPIRE *with built-in decision procedures before passing it to the underlying first-order theorem prover.*

The result of the preprocessing for Property 1 is

$$\begin{aligned}&\forall i,j.\left|\mathsf{verify}\begin{pmatrix}\pi_2(\mathrm{input}(\mathsf{Rs}[i,j])),\\ \pi_1(\mathrm{input}(\mathsf{Rs}[i,j]))\end{pmatrix}, \mathsf{k}[j]\right| \Rightarrow\\ &\quad\exists i',j'.\begin{pmatrix}\mathsf{T}[i',j'] < \mathsf{Rs}[i,j] \wedge j' = j\\ \wedge |\mathsf{n}[i',j']| = |\pi_1(\mathrm{input}(\mathsf{Rs}[i,j]))|\end{pmatrix}\end{aligned} \tag{5}$$

In (5)*, $\mathsf{T}[i',j'] < \mathsf{Rs}[i,j]$ and $j' = j$ appear as a consequence of the symbolic analysis. The variable u from* (2) *gets inlined.*

Remark 1 (Eliminating subterm reasoning)**.** *Note that the need for subterm reasoning is entirely eliminated from* (5)*. When the preprocessing is sufficiently comprehensive to* *establish the proof of the query without the help of the general axioms, we can factor out a significant portion of the subterm reasoning to a point that we can again reason modulo $\doteq$ while avoiding the unsoundness problems shown in Example 8. This heuristic, further described in Section 5.3.2, leads thus to a significant performance improvement, as shown in Section 6, at the cost of completeness.*

3. Preliminaries

This section delves into CRYPTOVAMPIRE's reasoning capabilities, which are grounded in an extension of the BC Logic ($\mathscr{BC}$) closely aligned with the one adopted in [19], [20]. We call this extension the "Symbolic Logic" ($\mathscr{S}$).

Section 4 later motivates and introduces our main contribution in the form of the "Evaluated Logic" ($\mathscr{E}$) that CRYPTOVAMPIRE uses to interact with $\mathscr{S}$. We show how we can leverage classical FOL methods to produce results in its specific semantics, and how these semantics closely match the BC Logic's and SQUIRREL's (resp. Property 5 and Theorem 1).

Notations. *We will use throughout this paper vector notations $\vec{y}$ as a shorthand for $y_1, \dots, y_n$ when n is obvious from the context. We use $\|\_\|$ to denote the size of a (finite) set or the length of a vector.*

We write $\mathrm{fv}(\phi)$ for the free variables of ϕ.

We also write the inference rule $\dfrac{A_1 \dots A_n}{B}$, *with $n \geq 0$, to mean $A_1 \wedge \dots \wedge A_n \Rightarrow B$ is valid.*

3.1. The BC Logic

CRYPTOVAMPIRE supports protocols and queries expressed in an extension of the BC Logic [15]. In this subsection, we give a quick introduction to the latter's syntax and semantics. We assume the following sets:

1) $\mathcal{N}_{\mathrm{bc}}$, a finite set of *nonce* names
2) $\mathcal{F}_{\mathrm{bc}}$, a finite set of *honest* functions
3) $\mathcal{G}_{\mathrm{bc}}$, a finite set of *attacker* functions

Terms in a BC Logic $\mathscr{BC}(\mathcal{N}_{\mathrm{bc}}, \mathcal{F}_{\mathrm{bc}}, \mathcal{G}_{\mathrm{bc}})$ are terms built from the following grammar:

$$t \coloneqq x \mid \mathsf{n} \mid f(\vec{t}) \mid g(\vec{t}) \tag{6}$$

with $\mathsf{n} \in \mathcal{N}_{\mathrm{bc}}$, $f \in \mathcal{F}_{\mathrm{bc}}$ and $g \in \mathcal{G}_{\mathrm{bc}}$; x is a variable.

Remark 2 (Connectives)**.** *Note $\mathscr{BC}(\mathcal{N}_{\mathrm{bc}}, \mathcal{F}_{\mathrm{bc}}, \mathcal{G}_{\mathrm{bc}})$ does not include the usual boolean connectives. To emulate them, we assume that $\mathcal{F}_{\mathrm{bc}}$ contains at least $\_ \doteq \_$,* IF _ THEN _ ELSE _, `true` *and* `false`. *We can then build the rest of the boolean connectives on top of these functions. To distinguish them from regular first-order boolean operators, we will overline them (e.g., we write $\bar{\wedge}$ instead of $\wedge$).*

A computational model $\mathbb{M}$ is an assignment of the elements of $\mathcal{N}_{\mathrm{bc}}$, $\mathcal{F}_{\mathrm{bc}}$, and $\mathcal{G}_{\mathrm{bc}}$ to polynomial-time Turing machines, which determine an interpretation $[\![t]\!]^{\mathbb{M}}_{\mathsf{bc}}$ of each BC term t. Specifically, this is defined as a polynomial-time Turing machine taking the security parameter η in unary and a pair $\rho = (\rho_h, \rho_{\mathcal{A}})$ of random tapes, one for honest agents and one for the adversary, respectively, constructed as follows:

1) $[\![\mathsf{n}]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, (\rho_h, \rho_{\mathcal{A}}))$, where $\mathsf{n} \in \mathcal{N}_{\mathrm{bc}}$, is a slice of length η of ρ_h, pairwise distinct for all elements of $\mathcal{N}_{\mathrm{bc}}$.
2) for $f \in \mathcal{F}_{\mathrm{bc}}$ we have
$$[\![f(t_1, \dots, t_n)]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, \rho) = [\![f]\!]\Big(1^\eta, [\![t_1]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, \rho), \dots, [\![t_n]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, \rho)\Big)$$
where $[\![f]\!]$ is a polynomial-time Turing machine with no direct access to ρ.
3) for $g \in \mathcal{G}_{\mathrm{bc}}$ we have
$$[\![g(t_1, \dots, t_n)]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, \rho) = [\![g]\!]\Big(1^\eta, \rho_{\mathcal{A}}, [\![t_1]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, \rho), \dots, [\![t_n]\!]^{\mathbb{M}}_{\mathsf{bc}}(1^\eta, \rho)\Big)$$
where $\rho = (\rho_h, \rho_{\mathcal{A}})$ and $[\![g]\!]$ is a polynomial-time Turing machine with direct access to $\rho_{\mathcal{A}}$ but not ρ_h. Thus terms of this form model attacker computations.

Property 2 (Subterm)**.** *For all models $\mathbb{M}$ and all η, t' appears in t iff, for all ρ, the Turing machine $[\![t]\!]^{\mathbb{M}}_{bc}$ applied to $(1^\eta, \rho)$ calls $[\![t']\!]^{\mathbb{M}}_{bc}$ on $(1^\eta, \rho)$.*

The $\mathscr{BC}$ is generally used with a somewhat peculiar notion of satisfiability: a formula holds when it evaluates to 1 with overwhelming probabilities. Formally:

Definition 1 (Cryptographic Satisfiability)**.** *We say that $\mathbb{M}$* satisfies *a BC formula t and write $\mathbb{M} \vDash_{\mathbb{P}} t$ when*

$$\mathbb{P}\mathrm{rob}_\rho\Big([\![t]\!]^{\mathbb{M}}_{bc}(1^\eta, \rho) \neq 1\Big) = \mathrm{negl}(\eta) \tag{7}$$

Where $\mathrm{negl}(\eta)$ *is a function h* negligible *in η, that is $h(\eta) = o(\eta^{-c})$ for all $c \in \mathbb{N}$.*

We say that a formula t is *cryptographically valid* for a set $\mathcal{C}_{\mathrm{bc}}$ of models and write $\mathcal{C}_{\mathrm{bc}} \vDash_{\mathbb{P}} t$ when it holds for all computational models $\mathbb{M}$ of $\mathcal{C}_{\mathrm{bc}}$.

The key idea of the CCSA is to reason within the biggest class $\mathcal{C}_{\mathrm{bc}}$ of models that respect some cryptographic assumptions about $\mathcal{F}_{\mathrm{bc}}$ and $\mathcal{G}_{\mathrm{bc}}$ (e.g., EUF-CMA) that we show are consistent with a computational attacker.

3.2. Modeling Protocols – The Symbolic Logic

Thus far, we have shown how to reason about symbolic computation. This is enough to describe a single protocol execution [15]: at any point, we can model the current message being sent as a BC term where the input is the list of all the messages sent before (we call it the *frame*) applied to an attacker function. However, our goal is to reason about all possible protocol executions. Hence, we illustrate now how to model protocols and the resulting extension of the base logic. We do so by closely following the formalism adopted in SQUIRREL [19], which makes CRYPTOVAMPIRE interoperable with it (Theorem 1).

We extend the syntax of (6) with the key ingredients to model protocols, as formalized in Fig. 1.

$$I := i \mid \mathbf{i}$$
$$T := \tau \mid \text{pred}(T) \mid \mathsf{a}[\vec{I}]$$
$$A := \text{happens}(T) \mid T < T' \mid T = T' \mid I = I'$$
$$t := x \mid \mathsf{n}[\vec{I}] \mid f[\vec{I}](\vec{t}) \mid \mathfrak{c}[\vec{I}]!(T) \mid A \mid \text{input}(T)$$
$$\mid \textsc{Find}\ \vec{\imath}\ \textsc{Such that}\ t\ \textsc{Then}\ t'\ \textsc{Else}\ t''$$
$$\mid \textsc{Find}\ \vec{\tau}\ \textsc{Such that}\ t\ \textsc{Then}\ t'\ \textsc{Else}\ t''$$

Figure 1: Grammar of $\mathbb{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$

Indices. We use indices to extend our reasoning to unbounded numbers of objects. Formally, indices are members $\mathbf{i}$ of a countable set $\mathcal{I}$.

Steps. Timepoints (or *steps*) are ranged over by T and represent atomic input/output operations in a protocol. We assume a finite set $\mathcal{S}$ of step names, containing at least the initialization step init $\in \mathcal{S}$. Steps are referenced by their name $\mathsf{a} \in \mathcal{S}$ and some indices, or relatively according to their order of execution, with $\text{pred}(T)$ denoting the step executed before T.

Predicates over protocol executions. The set of logical predicates characterizing a specific protocol execution, ranged over by A, includes $\text{happens}(T)$, which expresses whether or not the current execution includes T; $T < T'$, which specifies if T is executed before T', and $T = T$, which captures equality between steps.

Input. We abstract away the attacker functions with an $\text{input}(T)$ constructor. This represents which input the attacker gave to step T. It is an attacker function applied to the knowledge the attacker has gained until T's execution.

Memory cells. $\mathfrak{c}[\vec{I}]!(T)$ with $\mathfrak{c} \in \mathcal{C}$ represents what T has stored into the memory $\mathfrak{c}[\vec{I}]$. $\mathcal{C}$ is assumed to be finite.

Lookups. The Find _ Such that _ Then _ Else _ are lookups constructions. They incidentally also let us build quantifier-like objects.

$$\bar{\exists}\alpha.\ t := \begin{array}{l}\textsc{Find}\ \alpha\ \textsc{Such that}\ t\\ \quad \textsc{Then}\ \mathtt{true}\ \textsc{Else}\ \mathtt{false}\end{array} \tag{8}$$

$$\bar{\forall}\alpha.\ t := \bar{\neg}\bar{\exists}\alpha.\ \bar{\neg}t \tag{9}$$

Finally, i, τ and x are variables. The resulting logic is $\mathbb{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$.

Definition 2 (Step)**.** *A* step $\mathsf{a}[\vec{\imath}]$ *is an atomic input/output honest operation in a protocol. It is a triple* $(c[\vec{\imath}], m[\vec{\imath}], u_{\mathsf{a}[\vec{\imath}]})$ *composed of:*

1) *a term* $c[\vec{\imath}]$ *(referred to by* $\text{cond}(\mathsf{a}[\vec{\imath}])$*), describing the condition under which the step may be executed.*
2) *a term* $m[\vec{\imath}]$ *(referred to by* $\text{msg}(\mathsf{a}[\vec{\imath}])$*), describing the output of the step.*
3) *a function* $u_{\mathsf{a}[\vec{\imath}]}$ *from* $\{\mathfrak{c}[\vec{\jmath}] | \mathfrak{c} \in \mathcal{C}\}$ *to the terms such that* $\text{fv}(u_{\mathsf{a}[\vec{\imath}]}(\mathfrak{c}[\vec{\jmath}])) \subseteq \vec{\imath} \cup \vec{\jmath}$*. This function is independent from the indices, i.e., for all permutation* σ*,* $u_{\mathsf{a}[\sigma(\vec{\imath})]}(\mathfrak{c}[\sigma(\vec{\jmath})]) = \sigma(u_{\mathsf{a}[\vec{\imath}]}(\mathfrak{c}[\vec{\jmath}]))$*.*

None of these elements may use ground indices.

Given a set of step names $\mathcal{S}$, we write $\mathcal{S}_{\mathcal{I}}$ for the set of instantiated steps: $\{\mathsf{a}[\vec{\mathbf{i}}] | \mathsf{a}[\_] \in \mathcal{S}, \vec{\mathbf{i}} \in \mathcal{I}^{\|\vec{\mathbf{i}}\|}\}$. We are now ready to define protocols as follows.

Definition 3 (Protocol)**.** *A* protocol $\mathcal{P}$ *is a tuple* $(\mathcal{S}, \prec)$ *respectively composed of a set of* steps *names and a preorder over* $\mathcal{S}_{\mathcal{I}}$ *such that:*

1) init *is the smallest element of* $\mathcal{S}_{\mathcal{I}}$ *according to* $\prec$*.*
2) $\prec$ *is independent from the indices. That is for all indices* $\vec{\mathbf{i}}$*,* $\vec{\mathbf{j}}$ *and permutation* σ *of* $\mathcal{I}$*,* $\mathsf{a}[\vec{\mathbf{i}}] \prec \mathsf{b}[\vec{\mathbf{j}}]$ *iff.* $\mathsf{a}[\sigma(\vec{\mathbf{i}})] \prec \mathsf{b}[\sigma(\vec{\mathbf{j}})]$*.*
3) *Terms of the from* T *in a step* $\mathsf{a}[\vec{\mathbf{i}}]$ *may only be of the form* $\text{pred}^n(\mathsf{b}[\vec{\mathbf{j}}])$ *where* $\mathsf{b}[\vec{\mathbf{j}}] \preceq \mathsf{a}[\vec{\mathbf{i}}]$ *and* $n \geq 0$ *(i.e., steps may only reference previous steps).*
4) *Memory cell assignments must terminate (i.e., no cyclic call graphs are allowed).*

Example (5)**.** *We recall here Example 5 which describes the steps of Example 1:*

$\underline{\mathsf{T}[i,j]}$*: ($i^{th}$ execution of the j^{th} tag)*
Condition*:* `true`
Message*:* $\langle \mathsf{n}[i,j], \mathcal{H}(\mathsf{n}[i,j], \mathsf{k}[j]) \rangle$

$\underline{\mathsf{Rs}[i,j]}$*: (Successful authentication of the j^{th} tag on the i^{th} execution)*
Condition*:* $\mathsf{verify}(\pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j])$
Message*:* ok

where in *stands for* $\text{input}(\mathsf{Rs}[i,j])$*.*

$\underline{\mathsf{Rf}[i]}$*: (No authentication on the i^{th} execution)*
Condition*:* $\bar{\neg}\bar{\exists}j.\ \mathsf{verify}(\pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j])$
Message*:* ko

where in *stands for* $\text{input}(\mathsf{Rf}[i])$*.*

Example (6)**.** *We remind Example 6 describing the ordering of Example 1 as mutual exclusion.*

$$\textit{for all } i,\ j \neq k,\ (\mathsf{Rf}[i] \diamond \mathsf{Rs}[i,j]) \textit{ and } (\mathsf{Rs}[i,j] \diamond \mathsf{Rs}[i,k]) \tag{3}$$

Remark 3. *Mutual Exclusion We model the mutual exclusion predicate* $\mathsf{a}[\vec{\mathbf{i}}] \diamond \mathsf{b}[\vec{\mathbf{j}}]$ *from Example 6 as* $\mathsf{a}[\vec{\mathbf{i}}] \prec \mathsf{b}[\vec{\mathbf{j}}]$ *and* $\mathsf{b}[\vec{\mathbf{j}}] \prec \mathsf{a}[\vec{\mathbf{i}}]$ *(remember,* $\preceq$ *is just a preorder, antisymmetry is not required).*

Example 10. *Example 5 and Example 6 (reminded above) fully describe Example 1.*

Each execution of a protocol yields a trace. Intuitively, a trace is simply a chosen sequence of instantiated steps that does not contradict the protocol. Formally, we also consider a finite set of indices required to execute that trace:

Definition 4 (Trace)**.** *A* trace $\mathbb{T}$ *of a protocol is a tuple* $(\mathbb{I}, \mathbb{S}, \sigma_{\mathbb{I}}, \leq_{\mathbb{S}})$ *such that*

1) $\mathbb{I} \subset \mathcal{I}$ *is finite;*
2) $\mathbb{S} \subseteq \mathcal{S}_{\mathbb{I}} := \{\mathsf{a}[\vec{\mathbf{i}}] | \mathsf{a} \in S, \vec{\mathbf{i}} \in \mathbb{I}^{\|\vec{\mathbf{i}}\|}\}$ *contains* init*;*
3) $\leq_{\mathbb{S}}$ *is a total ordering over* $\mathbb{S}$ *compatible with* $\preceq$*, i.e., if* $\mathsf{a}[\vec{\mathbf{i}}] \preceq \mathsf{b}[\vec{\mathbf{j}}]$ *then* $\mathsf{a}[\vec{\mathbf{i}}] \leq_{\mathbb{S}} \mathsf{b}[\vec{\mathbf{j}}]$*;*
4) $\sigma_{\mathbb{I}} : \mathcal{I} \to \mathbb{I}$ *is the identity over* $\mathbb{I}$*.*

Remark 4. *Note that Remark 3 encodes the expected behavior of mutual exclusion: the compatibility of* $\leq_{\mathbb{S}}$ *ensures* *that* $\mathsf{a}[\vec{\mathbf{i}}]$ *and* $\mathsf{b}[\vec{\mathbf{j}}]$ *cannot be both in* $\mathbb{S}$*. Furthermore,* $\mathbb{S}$ *can be selected independently of the conditions of the steps. Instead, the failure of a step is modeled by returning* fail *to all subsequently scheduled steps.*

Example 11. *Reusing Example 1, let us assume we have two tags and two rounds. We can split the protocol into the steps* init, $\mathsf{T}[i,j]$, $\mathsf{Rs}[i,j]$ *and* $\mathsf{Rf}[i]$ *like in Example 5. Taking* $\mathcal{I} = \mathbb{N}$*, consider the following:*

$$\mathcal{S}_1 = \{\mathsf{init} <_1 \mathsf{T}[1,2] <_1 \mathsf{Rs}[1,2]\} \tag{10}$$

$$\mathcal{S}_2 = \{\mathsf{init} <_2 \mathsf{T}[1,1] <_2 \mathsf{Rs}[1,2] <_2 \mathsf{T}[2,2]\} \tag{11}$$

$$\mathcal{S}_3 = \{\mathsf{init} <_3 \mathsf{Rf}[1] <_3 \mathsf{T}[1,1] <_3 \mathsf{Rs}[1,1]\} \tag{12}$$

(10) *describes a valid trace with* $\mathbb{I} = \{1,2\}$*, and so is* (11) *despite step* $\mathsf{Rs}[1,2]$ *failing. However,* (12) *is not a valid trace as it contradicts* $\prec$ *despite the condition of all the steps seemingly holding.*

Definition 5 (Unfolding)**.** *Given a trace* $\mathbb{T}$*, we can instantiate and unfold any term* t *into a BC Logic term. We note* $[t]^{\mathbb{T}}$ *the result of this transformation.* $[t]^{\mathbb{T}}$ *follows closely [19]'s corresponding transformation and we formalize it in our setting in Appendix A.*

This unfolding turns any term into its BC equivalent for a given trace following the intuition given at the beginning of this section. For instance, $\mathrm{input}(T)$ becomes att applied to everything sent over the network before T.

Example 12. *Continuing Example 1, let us unfold* $\mathrm{input}(\mathsf{Rs}[1,2])$ *according to the trace* $\mathbb{T}_{(10)}$ *described in Equation 10 from Example 11:*

$$[\mathrm{input}(\mathsf{Rs}[1,2])]^{\mathbb{T}_{(10)}} = \mathsf{att}\left(\begin{array}{r} \ll [\mathrm{cond}(\mathsf{T}[1,2])]^{\mathbb{T}_{(10)}} \mathbin{\bar{\wedge}} [\mathrm{cond}(\mathsf{init})]^{\mathbb{T}_{(10)}}; \\ \ll \textsc{If}\ [\mathrm{cond}(\mathsf{T}[1,2])]^{\mathbb{T}_{(10)}} \mathbin{\bar{\wedge}} [\mathrm{cond}(\mathsf{init})]^{\mathbb{T}_{(10)}} \\ \textsc{Then}\ [\mathrm{msg}(\mathsf{T}[1,2])]^{\mathbb{T}_{(10)}}\ \textsc{Else}\ \mathsf{fail}; \\ [\mathrm{msg}(\mathsf{init})]^{\mathbb{T}_{(10)}} \gg\gg \end{array}\right) \tag{13}$$

This minimal example already shows that the attacker has access to the messages of all the previous steps and their condition. We also see that failing a condition blocks the execution of all subsequent steps as they return fail. Finally, Example 12 highlights the highly recursive nature of $[\_]^{\mathbb{T}}$.

4. CryptoVampire Formalization: A First-Order Theory of Protocol Queries

We now describe how to encode the Symbolic Logic from Section 3.2 in FOL in order to automate security proofs, and how to recover BC-like semantics from such a proof. This is particularly challenging, since several axioms of the Symbolic Logic, as well as the original BC Logic, rely on beyond what can be finitely axiomatized within the logic itself (Section 4.1.1). Moreover, the semantics of the BC Logic do not match that of classical FOLs, as assumed by first-order theorem provers (Section 4.1.2). In order to avoid higher-order and non-classical reasoning, which would degrade performance, we introduce a tailored encoding in FOL (Sections 4.2 and 5.1) as well as properties (Section 4.3) to ensure the soundness of the result.

4.1. The Challenges of Protocol Queries

4.1.1. Encoding challenges. Many cryptographic properties are defined in a way that is not finitely axiomatizable in the BC Logic, nor in our current Symbolic Logic. This motivates the need for an Evaluated Logic. In this subsection, we will look at the challenges that this new logic is meant to overcome. We illustrate them with the Euf-Cma property (cf. Property 1) that we state in its pure BC Logic form in Property 3.

Property 3 (Euf-Cma)**.** *Let* $\mathcal{C}_{\text{EUF-CMA}}$ *be a class of models where* $\mathcal{H}(\_,\_)$ *and* $\mathsf{verify}(\_,\_,\_)$ *form a* Euf-Cma*-secure* MAC *scheme. In the BC Logic (Section 3.1), we have for every signature* σ*, message* m *and key* k*:*

$$\mathcal{C}_{\text{EUF-CMA}} \vDash_{\text{bc}} \mathsf{verify}(\sigma, m, \mathsf{k}) \mathrel{\dot{\Rightarrow}} \dot{\bigvee}_{\mathcal{H}(u,\mathsf{k}) \in \mathsf{st}_{\mathscr{BC}}(m,\sigma)} u \doteq m \tag{14}$$

where k *only appears in the position of the key in* m *and* σ*.* $\mathsf{st}_{\mathscr{BC}}(\vec{t})$ *is the set of subterms appearing in* $\vec{t}$*.*

Property 3 highlights two challenges:

- Equation 14 is an axiom scheme. We cannot express it within the Symbolic Logic with finitely many axioms. (A)
- Property 3 also makes heavy use of subterm reasoning. Not only is this not finitely axiomatizable [28], even an incomplete finite axiomatization is beyond the Symbolic Logic due to unfortunate interactions with the equational theory. (B)

(A) Quantification over arbitrary terms. Equation 14 represents a set of axioms ranging over every term m, σ, and nonces k. Unfortunately, there are infinitely many terms. This can be solved with quantification: universal quantification over m, σ, and k and existential quantification over u to avoid having a disjunction that depends on m, σ, and k.

Yet, quantification over arbitrary terms is beyond the Symbolic Logic. Indeed, we can quantify over indices and timepoints only because we can unfold quantifiers into finitely nested If _ Then _ Else _ statements for any given trace (see Appendix A). Using the same trick with arbitrary terms would yield infinitely large terms. Even somehow expanding the Symbolic Logic to include such terms would contradict their interpretations into the BC Logic. Indeed, the interpretation $[\![t]\!]^{\mathbb{M}}_{\text{bc}}$ of a term t *must* be a *polynomial* time Turing machine.

We devise the Evaluated Logic to release ourselves from this last constraint. Thus, we include terms whose interpretation is beyond the BC Logic. We then show that this new interpretation not only matches BC's on common terms (Property 5) but expands it to any other terms (Theorem 3).

(B) Subterm Relation $\sqsubseteq$ and Equalities $\doteq$. In the BC Logic, honest and malicious computations are first described symbolically. This symbolic description informs on whether terms were directly computed or not. For instance, if a BC term t' does *not* appear in another BC term t (we write $t' \not\sqsubseteq t$), then we know that the Turing machine $[\![t]\!]^{\mathbb{M}}_{\mathsf{bc}}$ does *not* call $[\![t']\!]^{\mathbb{M}}_{\mathsf{bc}}$, as stated in Property 2. This allows us to closely capture the semantics of cryptographic games, and is already exposed in Property 3.

Example 13 (EUF-CMA)**.** *Within the* EUF-CMA *axiom of Property 3, the subterm property side condition on* k *guards against key misuses:* k *can only be used to hash and verify. Furthermore, the subterm property* $\mathcal{H}(u, \mathsf{k}) \in \mathsf{st}_{\mathscr{BC}}(m, \sigma)$ *guards against illegal uses of the signing oracle in the* EUF-CMA *cryptographic game: the attacker may ask to sign anything with* k *except for any term that evaluates as* m*.*

Let us look in more detail at the challenges associated with this subterm relation. Remember that in Example 8 we saw that the interplay between equality and subterm reasoning can easily lead to unsoundness. Example 14 shows how a naive encoding is unsound:

Example 14. *Consider now the subterm axiom*

$$x \sqsubset f(t_1, \ldots, t_n) \Leftrightarrow \bigvee_{i=1}^{n} x \sqsubseteq t_i. \tag{15}$$

Equation 15 is unsound *modulo* $\doteq$ *when tuples, projections, and empty are defined. It is indeed enough to implement Example 8 and thus show* $x \sqsubset \varnothing$*. However, using the* $\Rightarrow$ *implication of* (8)*, instantiated with* $\varnothing$*, we also derive* $x \not\sqsubset \varnothing$*, yielding thus a source of unsoundness.*

Unlike [29], we cannot circumvent the issue by relying on the specifics of $\doteq$ as it is a general equivalence relation, nor can we exploit specific solver decision procedures as we aim for a sound first-order encoding. Instead, we go around the issue in our Evaluated Logic by sandboxing the reasoning modulo $\doteq$ with the evaluation predicate $|\_|$. We keep $=$ as the symbolic equality because we have $(x = y) \Rightarrow |x \doteq y|$. Since $|\_ \doteq \_|$ is an equivalence relation, we can let theorem provers use their powerful equality reasoning with $\doteq$ by introducing the sugar:

$$|t| = |t'| := |t \doteq t'| \tag{16}$$

We recover some of the overhead introduced by $|\_|$ by axiomatizing the commutativity of $|\_|$ with boolean connectives (Property 4). The concrete axioms are listed in Fig. 8 of Appendix B.

All in all, we address the above challenges by embedding the Symbolic Logic of Section 3.2 into an *Evaluated Logic* where we can safely quantify over arbitrary terms and reason symbolically about subterms. Section 4.2 presents the syntax and semantics of this new logic, while Section 5.1 dives deeper into our modeling of subterm analysis.

4.1.2. Soundness challenges. As already noted in [15], the BC Logic's semantics does not match that of classical FOL. Indeed, the notion of satisfiability introduced in Definition 1 quickly gives us the intuition that a term can be true, false, or something in between.

$$\begin{aligned} t &:= \ldots \\ S &:= t = t' \,\big|\, t \sqsubseteq t' \,\big|\, \ldots \\ \varphi &:= \top \,\big|\, \bot \,\big|\, S \,\big|\, |t| \,\big|\, \varphi \vee \varphi' \,\big|\, \neg\varphi \\ &\quad \big|\, \exists \vec{\imath}, \vec{\tau}, \vec{x}.\varphi \,\big|\, \forall \vec{\imath}, \vec{\tau}, \vec{x}.\varphi \end{aligned}$$

where t is a term from $\mathscr{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ as defined in Fig. 1. We extend with $\wedge$, $\Rightarrow$ and $\Leftrightarrow$ as expected, and write $|t| = |t'|$ as sugar for $|t \doteq t'|$.

Figure 2: The Evaluated Logic $\mathscr{E}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$

$$\mathcal{L}^{\eta}(|t|)(\rho) := \min\left(1, \left[\!\!\left[[t]^{\mathbb{T}}\right]\!\!\right]^{\mathbb{M}}_{\mathsf{bc}}(1^{\eta}, \rho)\right) \tag{17}$$

$$\mathcal{L}^{\eta}(t \sqsubseteq_R t')(\rho) := \mathcal{M}(\sqsubseteq_R)(t, t') \tag{18}$$

$$\mathcal{L}^{\eta}(t = t')(\rho) := \mathcal{M}(=)(t, t') \tag{19}$$

Where $\mathcal{M}(=)$ behaves like an equivalence relation.
The boolean connectives and quantifiers behave assuming $\mathcal{L}^{\eta}(\_)(\rho)$ is a FOL model.

Figure 3: Cryptographic Model

Example 15 (Counterexample to classical semantics)**.** *Let* $\mathbb{M}$ *be a computational model that interprets the function* $1^{\text{st}}\ \text{bit}(x)$ *as the first bit of* x*. Let* n *be a nonce; we get that* $\mathbb{M} \vDash_{\mathbb{P}} (1^{\text{st}}\ \text{bit}(\mathsf{n}))\ \dot{\vee}\ \dot{\neg}(1^{\text{st}}\ \text{bit}(\mathsf{n}))$ *as it is always true. However, we have neither* $\mathbb{M} \vDash_{\mathbb{P}} (1^{\text{st}}\ \text{bit}(\mathsf{n}))$ *nor* $\mathbb{M} \vDash_{\mathbb{P}} \dot{\neg}(1^{\text{st}}\ \text{bit}(\mathsf{n}))$ *as they both alternate between* 0 *and* 1 *with probability* $\frac{1}{2}$*.*

The upcoming Evaluated Logic is designed to fix the issues introduced in Examples 14 and 15.

4.2. Evaluated Logic

Let us call Ω the set of random tape pairs. $\mathbb{P}\text{rob}_{\rho}(\_)$ is a probability measure over Ω.

4.2.1. Syntax. The Evaluated Logic, denoted as $\mathscr{E}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$, extends the Symbolic Logic $\mathscr{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ from Section 3.2 with predicates over terms. In a nutshell, the Evaluated Logic is a standard FOL whose literals are predicates over terms from the Symbolic Logic, or evaluation of those terms, denoted by the predicate $|\_|$. The evaluation of a term forces its computational interpretation.

The syntax is described in Fig. 2: $t \sqsubseteq t'$ expresses that t is a subterm of t', $|t|$ captures the concrete evaluation of the symbolic term t (e.g., symbolic cryptographic functions are evaluated into concrete bitstrings); φ ranges over first-order formulas with existential and universal quantification over terms.

4.2.2. Semantics.

Definition 6 (Cryptographic Model)**.** *A cryptographic model $\mathcal{L}$ is a tuple $(\mathbb{M}, \mathbb{T}, \mathcal{M})$ where $\mathbb{M}$ is a computational model, $\mathbb{T}$ a trace, and $\mathcal{M}$ a symbolic model.*

$\mathcal{L}$ associates terms of $\mathscr{E}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ to functions of $\mathbb{N} \times \Omega \to \{0, 1\}$ according to Fig. 3. We write $\mathcal{L}(\varphi)$ the interpretation of φ by $\mathcal{L}$ and $\mathcal{L}^\eta(\varphi)$ to express the random variable $[\rho \in \Omega \mapsto \mathcal{L}^\eta(\varphi)(\rho)]$.

Remark 5. *As compared to the semantics adopted in Squirrel [20], we avoid the probabilistic interpretation of symbolic terms. That is, the interpretation of $|t|$ is the underlying Turing machine* applied *to some η and ρ, and not $\vDash_{\mathbb{P}} (t)^{\mathbb{T}}$ as seen in previous works [20].*

Remark 5 solves Example 15 sufficiently for our trace properties and allows us to move boolean connectives between logics, greatly improving the performance of theorem provers (Property 4). Achieving such a functionality for indistinguishability properties is more challenging [18] and would require dedicated interpretations of the evaluation predicate.

Property 4. *$|\_|$ commutes with boolean connectives and quantifiers.*

Proof: Via Property 9 from Appendix B. □

4.2.3. Relation to the BC Logic and SQUIRREL.

We now intuitively explain how we relate this semantic interpretation to the one of the BC Logic (Definition 1).

Definition 7 (Cryptographic Satisfiability)**.** *We adapt Definition 1 to the new setting. Thus $\mathcal{L}$* cryptographically satisfies *φ when*

$$\mathbb{P}\mathrm{rob}_\rho(\mathcal{L}^\eta(\varphi) \neq 1) = \mathrm{negl}(\eta) \tag{20}$$

and we reuse the notation $\mathcal{L} \vDash_{\mathbb{P}} \varphi$ and also extend it to class $\mathbb{C}$ of cryptographic models like so: $\mathbb{C} \vDash_{\mathbb{P}} \varphi$.

We connect this notion of validity to the one of the BC Logic (Definition 1) as described in Property 5.

Property 5 (Extension of BC)**.** *Let t be a symbolic term. Its evaluation in $\mathscr{E}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ is* valid *iff any unfolding in the BC Logic is valid. Formally:*

$$\mathbb{C} \vDash |t| \text{ iff for all trace } \mathbb{T} \text{ we have } \mathbb{C}_{\mathbb{M}} \vDash_{\mathrm{bc}} [t]^{\mathbb{T}} \tag{21}$$

where $\mathbb{C}_{\mathbb{M}}$ are the computational models of $\mathbb{C}$.

We can in fact find a transformation $\mathscr{T}(\_)$ from a relevant subset of SQUIRREL's term to CRYPTOVAMPIRE's such that their interpretation coincide.

Intuitively, $\mathscr{T}(\_)$ commutes with every common symbol (e.g., functions, nonces, quantifiers,…). SQUIRREL's macros are either unfolded (e.g., $\mathsf{cond}@T$ is unfolded into the corresponding condition) or replaced by the corresponding CRYPTOVAMPIRE term (e.g., $\mathsf{input}@T$ is replaced by $\mathrm{input}(\mathscr{T}(\underline{T}))$). The remaining SQUIRREL terms have a direct counterpart in CRYPTOVAMPIRE. The only exception is the $\mathsf{frame}@T$ macro due to its recursive nature: this is, however, typically not used directly and is instead only used to define $\mathsf{input}@T$, which our transformation supports. While our transformation does not aim at completeness, all examples from Section 6 are supported.

Remark 6 (Quantifiers)**.** *The translation only preserves the interpretation, not the unfolding. This especially matters with quantifiers: they unfold differently, but they evaluate to logically equivalent disjunctions/conjunctions in both tools (see Lemma 2 of Appendix C and Property 4).*

Theorem 1 (Interoperability)**.** *For all* SQUIRREL *protocol $\mathcal{P}_{\mathrm{sq}}$, we can find a* CRYPTOVAMPIRE *protocol $\mathcal{P}$, such that for all* SQUIRREL *trace $\mathbb{T}_{\mathrm{sq}}$ over $\mathcal{P}_{\mathrm{sq}}$, we can find a* CRYPTOVAMPIRE *trace over $\mathcal{P}$, such that for all computational models $\mathbb{M}$, security parameter η, pairs of random tapes ρ, and* SQUIRREL *t over which $\mathscr{T}(\_)$ is defined, we have*

$$\mathcal{L}^\eta(|\mathscr{T}(\underline{t})|)(\rho) = \left[\!\left[[t]_{\mathrm{sq}}^{\mathbb{T}_{\mathrm{sq}}} \right]\!\right]_{bc}^{\mathbb{M}} (1^\eta, \rho) \tag{22}$$

where $[\_]_{\mathrm{sq}}^{\mathbb{T}_{\mathrm{sq}}}$ is the SQUIRREL *unfolding from [19], [20].*

For more details, we refer to Appendix E.

4.3. Linking Cryptographic Semantics and Classical First-Order Logic

In this section, we first show that the Evaluated Logic is not enough to faithfully encode the behavior of cryptographic protocols (Section 4.3.1), and we propose a method to sidestep the problem while retaining both the automation potential offered by classical FOL and the cryptographic semantics of Definition 7 (Section 4.3.2).

4.3.1. The Problem with cryptography-related axioms.

We first show that Definition 7 does not satisfy many critical axioms while, at the same time, allowing for some cryptographically unsound formulas. Consider Theorem 2 (adjusted from [15], [16], proven in Appendix F):

Theorem 2 (No Guessing)**.** *It is not possible to guess honest nonces. Formally, for all nonce $\mathsf{n}[\_] \in \mathcal{N}$, ground indices $\vec{\mathbf{i}} \in \mathcal{I}$, and message m, we have*

$$|\mathsf{n}[\vec{\mathbf{i}}]| = |m| \Rightarrow \mathsf{n}[\vec{\mathbf{i}}] \sqsubseteq m \tag{23}$$

Equation 23 describes that any message m that evaluates to a nonce $\mathsf{n}[\vec{\mathbf{i}}]$ must *symbolically* contain that nonce. This theorem is valid for all cryptographic models and is fundamental to many proofs as it encodes how randomness behaves in the logic. However, (23) is still an axiom scheme. Unfortunately, turning it into a formula breaks down as shown in Example 16.

Example 16 (The Evaluated Logic rejects Theorem 2)**.** *Any cryptographic model that can express the natural numbers cannot satisfy* (24). *(notice the $\forall$)*

$$\forall \vec{i}, x.\ |\mathsf{n}[\vec{i}]| = |x| \Rightarrow \mathsf{n}[\vec{i}] \sqsubseteq x \tag{24}$$

This is a consequence of the even worse Example 17.

Example 17 (The Evaluated Logic allows for unsound formulas)**.** *Assuming $\mathcal{F}$ contains* 0 *and* $S(\_)$*, any cryptographic model that interprets* 0 *and* $S(\_)$ *as respectively* 0 *and the successor function satisfies* (25).

$$\exists x.\ |\mathsf{n}[\vec{\mathbf{1}}]| = |x| \wedge \mathsf{n}[\vec{\mathbf{1}}] \not\sqsubseteq x \tag{25}$$

Proof: Let $\mathcal{L}$ be such model. $\mathcal{L}^\eta(\mathsf{n}[\vec{\mathbf{1}}])(\rho)$ is a natural number N. We then define $\underline{N}_S := S^N(0)$, where S^k represents k compositions of S. We get that $\mathcal{L}^\eta(\mathsf{n}[\vec{\mathbf{1}}])(\rho) = \mathcal{L}^\eta(\underline{N}_S)(\rho)$ and yet $\mathsf{n}[\vec{\mathbf{1}}] \not\sqsubseteq \underline{N}_S$. As this holds for any η and ρ, we get (25). □

The $\underline{N}_S$ that appears in the proof of Example 17 is a side effect of Remark 5. We can observe this side effect even more clearly when putting (25) in Skolem normal form [31]:

$$|\mathsf{n}[\vec{\mathbf{1}}]| = |\mathrm{sk}_{(25)}| \wedge \mathsf{n}[\vec{\mathbf{1}}] \not\sqsubseteq \mathrm{sk}_{(25)} \tag{26}$$

The resulting Skolem $\mathrm{sk}_{(25)}$ is beyond our control. For any η and ρ, $\mathrm{sk}_{(25)}$ may take the value of any constant that just so happens to evaluate like $\mathsf{n}[\vec{\mathbf{1}}]$ for this *specific* η and ρ. This makes it a *non-polynomial* function that guesses $\mathsf{n}[\vec{\mathbf{1}}]$.

In the next section, we show how to reject formulas that may produce such unwanted behavior while allowing for well-behaved ones like Equation 24.

4.3.2. Linking First-order and Cryptography Together.

Example 17 is a result of $\mathrm{sk}_{(25)}$ being able to take too many values. Thus, we say that a formula has a *bounded Skolem normal form* when we can find *finite* sets of terms such that we can safely assume that the Skolems will only take values in those sets. We formalize this notion in Definition 18 of Appendix C. In practice, we do not need to go back to the definition, and we use the following properties instead, yielding an almost syntactic description of the formulas verifying the property.

The base case is handled by Property 6, meaning that any quantifier-free formula, or with simple enough quantification, has a bounded Skolem normal form.

Property 6. *A formula whose Skolem normal form is already in* $\mathscr{L}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ *(i.e., no new symbols are required) has a bounded Skolem normal form.*

Then Property 7 and Property 8 let us build new formulas with bounded Skolem normal forms from existing ones. Definition 8 describes formulas that ensure terms cannot take too many values; consequently, they can prevent Skolem functions from displaying the behavior described in Section 4.3.1. They, therefore, allow existential quantification. Notably, subterm relations produce such formulas.

Property 7. *The notion of bounded Skolem normal form is stable by conjunction and disjunction.*

Definition 8 (Bounding Formula)**.** *We say a formula φ with* $\mathrm{fv}(\varphi) = x \uplus \vec{u}$ *is* bounding *on x if for all $\vec{\boldsymbol{v}}$ we can find a set $D_\varphi^{\vec{\boldsymbol{v}}}$ such that for all η, ρ, t, and cryptographic models $\mathcal{L}$, we have*

$$\mathcal{L}^\eta\left(\varphi\begin{Bmatrix} x \mapsto t \\ \vec{u} \mapsto \vec{\boldsymbol{v}}\end{Bmatrix} \Rightarrow \bigvee_{\substack{\phi \in D_\varphi^{\vec{\boldsymbol{v}}} \\ \vec{\alpha} = \mathrm{fv}(\phi)}} \exists \vec{\alpha}.\ \phi = t\right)(\rho) = 1 \tag{27}$$

Property 8. *If φ is bounding on x and has a bounded Skolem normal form, then $\exists x.\ \varphi$ has a bounded Skolem normal form.*

Finally, formulas with bounded Skolem normal forms enjoy a property close to the modus Polens:

Theorem 3 (Cryptographic Validity)**.** *Let φ be a formula s.t.* $\mathrm{fv}(\varphi) = \vec{u}$*, t a term, and $\mathbb{C}$ a set of cryptographic models such that*

1) *φ has a bounded Skolem normal form;*
2) *for all $\vec{\boldsymbol{v}}$, $\mathbb{C} \vDash_{\mathbb{P}} \varphi\{\vec{u} \mapsto \vec{\boldsymbol{v}}\}$;*
3) *$(\forall \vec{u}.\ \varphi) \Rightarrow |t|$ is valid in* FOL*;*

then $\mathbb{C} \vDash_{\mathbb{P}} |t|$.

Theorem 3 links the cryptographic semantics of axioms (2) with first-order reasoning (3) and extracts a cryptographic property so long as these axioms are well-behaved (1), i.e., they have bounded Skolem normal forms.

We notice now that property (24) of Example 16 verifies items (1) and (2). It is therefore usable in first-order proofs (and so is Theorem 2). Similarly, we can turn the axiom schema (2) of Property 1 to a formula that verifies (1) and (2). We prove the properties introduced in this section in Appendix C.

5. CryptoVampire Reasoning for Proving Protocol Queries

Based on the first-order formalization from Section 4, we now describe how to extend saturation-based first-order theorem proving [25] in order to achieve automation in a first-order setting. The main problem to be solved is how to efficiently encode subterm reasoning in FOL. Section 5.1 reviews the higher-order definition of subterm, which is similar to the one adopted in Squirrel [19]. Section 5.2 presents an efficient solving procedure grounded in first-order saturation that we embed in the Vampire theorem prover. Section 5.3 introduces preprocessing and heuristic techniques that further improve performance and that we integrate in CryptoVampire.

5.1. Subterm Relations in CryptoVampire

As explained in the sections above, the logics presented in this paper ground their semantics in the BC Logic, including subterm relations. In the BC Logic, the notion is fairly straightforward: for instance, the strict subterms of $\langle x, y\rangle$ are x and y (assuming x and y are constant symbols). However, this simplicity is lost after the unfolding $[\_]^{\mathbb{T}}$ of Definition 5. We explore now the problems that arise and our solutions.

Example (12)**.** *Recall Definition 5 and Example 12: let us unfold* input$(\mathsf{Rs}[1,2])$ *according to the trace $\mathbb{T}_{(10)}$ described* *in Equation 10 from Example 11:*

$$[\mathrm{input}(\mathsf{Rs}[1,2])]^{\mathbb{T}_{(10)}} = \mathsf{att}\begin{pmatrix} \ll [\mathrm{cond}(\mathsf{T}[1,2])]^{\mathbb{T}_{(10)}} \barwedge [\mathrm{cond}(\mathsf{init})]^{\mathbb{T}_{(10)}}; \\ \ll \text{IF } [\mathrm{cond}(\mathsf{T}[1,2])]^{\mathbb{T}_{(10)}} \barwedge [\mathrm{cond}(\mathsf{init})]^{\mathbb{T}_{(10)}} \\ \text{THEN } [\mathrm{msg}(\mathsf{T}[1,2])]^{\mathbb{T}_{(10)}} \text{ ELSE } \mathsf{fail}; \\ [\mathrm{msg}(\mathsf{init})]^{\mathbb{T}_{(10)}} \gg\gg \end{pmatrix} \tag{13}$$

A naive approach to implementing the BC subterm in CRYPTOVAMPIRE would be to axiomatize the unfolding of $[\_]^{\mathbb{T}}$. However, Example 12 highlights multiple problems: **(i)** the unfolding is dependent on the trace that we would like to abstract away, and **(ii)** it is highly recursive. Point (ii), in particular, would require some inductive reasoning, but induction is not a first-order property and is extremely challenging to algorithmically address [32].

Instead, reusing insights from [19], we encode this inductive reasoning with subterms directly into our description of a subterm relation. That is, we design a $\sqsubseteq$ relation that overapproximates the BC notion of subterm. This is achieved through a set $\mathsf{st}_{\mathcal{P}}(\_)$, which can be computed by CRYPTOVAMPIRE with the help of a theorem prover to solve problems (i)–(ii).

Concretely, $\mathsf{st}_{\mathcal{P}}(t)$ lists all the terms t' for which we can find a trace $\mathbb{T}$ such that t' unfolds into a BC subterm of the unfolding of t. To keep the number of plausible t' in check, we add a guard on the trace that unfolds to `false` in traces where t' is definitely not a subterm of t. Overall, we then define $\sqsubseteq$ as the following

Definition 9 (CRYPTOVAMPIRE's base subterm)**.** *We define the* base subterm *relation* $\sqsubseteq$ *on any term* t *as such:*

$$t \sqsubseteq t' \text{ iff } \bigvee_{\substack{(c,u)\in \mathsf{st}_{\mathcal{P}}(t') \\ \text{with } \vec{\imath}=\mathrm{fv}(u)\cup\mathrm{fv}(v)}} \exists \vec{\imath}.|c| \wedge u = t \tag{28}$$

where $\mathsf{st}_{\mathcal{P}}(\_)$ *is fully described in Appendix D.*

We prove in Appendix D how $\sqsubseteq$ is an overapproximation of the BC notion of subterm. We exemplify how $\mathsf{st}_{\mathcal{P}}(\_)$ is built in Example 18.

Example 18 (Subterm set)**.** *Let* $ST(\_)$ *be the syntactic subterms, that is, for instance:*

$$ST(\mathsf{verify}(\pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j])) := \{\mathsf{verify}(\pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j]), \pi_2(\mathsf{in}), \pi_1(\mathsf{in}), \mathsf{k}[j], \mathsf{in}\} \tag{29}$$

where in *stands for* $\mathrm{input}(\mathsf{Rs}[i,j])$ *as in Example 5.*

Then, with $\mathcal{P}_1$ *being the protocol of Example 1, keeping in mind its steps description (Example 5), we have*

$$\begin{aligned} \mathsf{st}_{\mathcal{P}_1}(\mathrm{input}(\tau)) := & \bigcup_{t\in ST(\mathrm{msg}(\mathsf{T}[i,j])\cup ST(\mathrm{cond}(\mathsf{T}[i,j])}(\mathsf{T}[i,j] < \tau, t) \\ \cup & \bigcup_{t\in ST(\mathrm{msg}(\mathsf{Rs}[i,j])\cup ST(\mathrm{cond}(\mathsf{Rs}[i,j])}(\mathsf{Rs}[i,j] < \tau, t) \\ \cup & \bigcup_{t\in ST(\mathrm{msg}(\mathsf{Rf}[i])\cup ST(\mathrm{cond}(\mathsf{Rf}[i])}(\mathsf{Rf}[i] < \tau, t) \end{aligned} \tag{30}$$

$$t \sqsubseteq t' := t \sqsubset t' \vee t = t' \tag{31}$$

$$\forall t, \vec{\imath}.\ t \not\sqsubset \mathsf{n}[\vec{\imath}] \tag{32}$$

$$\forall t, t_1, \dots, t_k, \vec{\jmath}.\ t \sqsubset f[\vec{\jmath}](t'_1, \dots, t'_k) \Rightarrow \bigvee_{i=1}^{n} t \sqsubseteq t'_i \tag{33}$$

$$\forall t, t_1, t_2, t_3.\ t \sqsubset \begin{pmatrix} \text{FIND } \vec{\alpha} \text{ SUCH THAT } t_1 \\ \text{THEN } t_2 \text{ ELSE } t_3 \end{pmatrix} \Rightarrow \exists \vec{\alpha}. \bigvee_{k=3}^{n} t \sqsubseteq t_k \tag{34}$$

$$\forall t, \tau.\ t \sqsubset \mathrm{input}(\tau) \Rightarrow \bigvee_{\substack{(c,u)\in \mathsf{st}_{\mathcal{P}}(\mathrm{input}(\tau)) \\ \text{with } \vec{\imath}=\mathrm{fv}(u)\cup\mathrm{fv}(c)}} \exists \vec{\imath}.|c| \wedge u = t \tag{35}$$

$$\forall t, \vec{\imath}, \tau.\ t \sqsubset \mathfrak{c}[\vec{\imath}]!(\tau) \Rightarrow \bigvee_{\substack{(c,u)\in \mathsf{st}_{\mathcal{P}}(\mathfrak{c}[\vec{\imath}](\tau)) \\ \text{with } \vec{\imath}=\mathrm{fv}(u)\cup\mathrm{fv}(v)}} \exists \vec{\jmath}.|c| \wedge u = t \tag{36}$$

Figure 4: Axiomatization of the subterm relation

With $\mathsf{st}_{\mathcal{P}}(\_)$ we can take the knowledge of the protocol and the inductive reasoning out of the theorem prover and internalize it into CRYPTOVAMPIRE itself. In practice, we describe $\sqsubseteq$ in FOL not as Equation 28 but through the axioms laid out in Fig. 4. With this axiomatization, we dispense from (i) the trace dependence as well as (ii) inductive reasoning. We describe how to extend $\sqsubseteq$ into other subterm relations (e.g., $\sqsubseteq_{\mathsf{verify}(\_,\_,\bullet),\mathcal{H}(\_,\bullet)}$) inAppendix G.2.

5.2. Native Subterm Reasoning

As the subterm relation is transitive, the number of logical consequences produced by saturation quickly becomes a burden for any saturation-based prover. While the $\mathsf{st}_{\mathcal{P}}(\_)$ set can effectively be computed (cf. [19] and Appendix D), automated reasoning with such sets is non-trivial due to their complex interaction with the equational theory of $\doteq$ and the required knowledge of the protocol. To overcome such limitations hindering the effectiveness of automation, we devise our own native subterm reasoning within saturation, extending existing approaches [28], [29].

We recall that saturation-based provers iteratively apply a finite set of *inference rules*, deriving new clauses[2] as logical consequences of existing formulas. Whenever the empty clause $\perp$ is inferred, saturation stops and reports unsatisfiability of the negated input formula (hence, the input is valid). For efficiency reasons, saturation implements two types of inferences: **(i)** *generating* rules that add new clauses to the search space, and **(ii)** *simplifying* rules that remove so-called redundant clauses from the search space. Importantly, removing redundant clauses does not destroy (refutational) completeness: if a formula is provable using redundant clauses, it is also provable without redundant clauses [25]. Clearly, simplifying rules are eagerly applied within any efficient saturation-based proving process. In what follows, within simplifying rules, we will denote the

[2] formulas are preprocessed and clausified

deleted (redundant) clause by drawing a line through it, for example: $\cancel{A \vee C}$.

The crux of our approach for native subterm reasoning within CryptoVampire comes with two new simplifying inference rules. Given a list $\mathfrak{L}$ of symbols and a subterm relation $\sqsubseteq$, for all $f \in \mathfrak{L}$, we introduce the following simplifying rules *in addition* to the superposition inference rules for FOL with equality [25]:

$$\frac{\cancel{x \sqsubseteq f(y_1, \ldots, y_n) \vee C}}{x = f(y_1, \ldots, y_n) \vee \bigvee_{k=1}^{n} x \sqsubseteq y_k \vee C}\text{Sbtm}$$

$$\frac{\cancel{x \not\sqsubseteq f(y_1, \ldots, y_n) \vee C}}{x \neq f(y_1, \ldots, y_n) \vee C \qquad x \not\sqsubseteq y_k \vee C \qquad \text{for all } k}\text{-Sbtm}$$

The inference rules Sbtm and -Sbtm replace axiomatic reasoning with the axioms (31) and (33) of Fig. 4. We prevent these rules from contradicting the rest of the axioms of Fig. 4 by selectively leaving out symbols from $\mathfrak{L}$; this is notably the case for input and memory cells.

The simplifying nature of Sbtm and -Sbtm ensures that m in $x \sqsubseteq m$ is fully destructed (up to special cases) before the conclusions are added to the search space.

5.3. Preprocessing

5.3.1. Instance preprocessing. While the $\mathsf{st}_{\mathcal{P}}(\_)$ sets, defining subterm relations, are computable, it is not known *a priori* over which terms $\mathsf{st}_{\mathcal{P}}(\_)$ should be applied. Hence, the axiomatization discussed in Section 5.1. Nevertheless, we may compute estimates about terms that may be involved in subterm reasoning, allowing us to predict/prioritize the application of subterm analysis, as follows.

Most cryptographic axioms have the pattern $|\phi| \Rightarrow \varphi$, where ϕ is an easy-to-identify symbolic term (e.g., $\mathsf{verify}(\sigma, m, \mathsf{k})$ for Euf-Cma in Property 1) and φ depends on some subterm analysis based on instances of variables in ϕ (e.g., $\mathcal{H}(u, \mathsf{k}) \sqsubseteq m$, among others, in Property 1 too). If we know ϕ, then we can often pre-compute φ to such an extent that it no longer contains any subterm search by inlining the subterm relation (Definition 9).

We therefore consider the following CryptoVampire heuristic: when we encounter a term that can unify with ϕ, we assume that it is likely to be used within proof search, through the respective cryptographic axiom. We then preprocess all occurrences of ϕ found throughout the protocol specification, including the step definitions, queries, and other user-defined assertions.

Example (9)**.** *Equation 5 from Example 9 is the result of such heuristic applied to Property 1 in Example 1.*

$$\begin{aligned}\forall i, j. \left| \mathsf{verify}\begin{pmatrix}\pi_2(\mathrm{input}(\mathsf{Rs}[i,j])), \\ \quad \pi_1(\mathrm{input}(\mathsf{Rs}[i,j]))\end{pmatrix}, \mathsf{k}[j] \right| \Rightarrow \\ \exists i', j'. \begin{pmatrix}\mathsf{T}[i',j'] < \mathsf{Rs}[i,j] \wedge j' = j \\ \quad \wedge |\mathsf{n}[i',j']| = |\pi_1(\mathrm{input}(\mathsf{Rs}[i,j]))|\end{pmatrix}\end{aligned} \tag{5}$$

5.3.2. Removing subterm reasoning. Finally, we propose an incomplete heuristic to completely factor out symbolic reasoning when applicable.

Generating terms that are not part of the input problem can, in general, lead to state explosion, but it is nevertheless necessary during proof search; for example, such terms might be needed to instantiate another cryptographic axiom. On the other hand, if an axiom instance does not get preprocessed in Section 5.3.1, then its instantiation will likely get significantly delayed during saturation[3]. For that reason, we propose to continue the preprocessing under the assumption that Section 5.3.1 effectively preprocessed *all* the relevant instances. Under such assumption, it is possible to factor out all symbolic reasoning (as it was inlined during preprocessing, as in Example 9). That is, when trying to prove $\vDash (\forall \vec{u}.\Gamma) \Rightarrow |t|$ for Theorem 3's (3), we can find a new axiom Δ (the result of the preprocessing) such that $\vDash (\forall \vec{u}.\Gamma) \Rightarrow \Delta$ and all symbols used in Δ are compatibles with $\doteq$. Then showing $\Delta \Rightarrow |t|$ can be done modulo $\doteq$, which dramatically improves the performance of the solvers.

Interestingly, this idea can be applied not only to protocol specifications but also some cryptographic theorems such as the no-guessing theorem. It can, in particular, be rephrased in a way to eliminate subterm reasoning.

Theorem 4 (No guessing)**.** *For all nonces* $\mathsf{n} \in \mathcal{N}$*,* (37) *can be added to the set of axioms.*

$$\forall \vec{\imath}, m.\ |\mathsf{n}[\vec{\imath}]| = |m| \Rightarrow \mathsf{n}[\vec{\imath}] \sqsubseteq^\circ m \tag{37}$$

where $\sqsubseteq^\circ$ *is defined by* (38) *and is compatible with* $\doteq$*.*

$$\mathsf{m} \sqsubseteq^\circ |t| \coloneqq \forall t'.(|t| = |t'| \Rightarrow \mathsf{m} \sqsubseteq t'). \tag{38}$$

Proof: Appendix F.1 □

While this heuristic is not complete, our experiments demonstrate that it is widely applicable and leads to significant performance gains (cf. Section 6).

6. Experiments

We implemented CryptoVampire in extension of the Vampire theorem prover [25], by adjusting saturation, term ordering, subterm reasoning and preprocessing in Vampire as described in Section 5.

Experimental Setup. We evaluated CryptoVampire against examples and trace-based queries from [19] that did not rely on the composition framework [22]. For competitive evaluations, we varied the proving backend using a selection of state-of-the-art solvers for FOL with theories, Z3 [24], cvc5 [26] and Vampire [25] (with and without our modifications), and used various levels of preprocessing.

Benchmarks. We test CryptoVampire on all trace-based properties from the Squirrel's benchmarks [19], which consist of authentication properties in the `basic-hash` [27], `hash-lock` [33], `lak-tag` [34],

[3]The machine is likely to run out of resources before that.

protocol		no preprocessing				instance preprocessing				Section 5.3.2's heuristic		
		Z3	CVC5	VAMPIRE ○	VAMPIRE ●	Z3	CVC5	VAMPIRE ○	VAMPIRE ●	Z3	CVC5	VAMPIRE
`basic-hash`	□/△	✗	✓	✗	✓	✓*	✓	✓	✓	✓	✓	✓
`hash-lock`	□	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
	■	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
	△	✗	✗	✗	✗	✗	✗	✗	✓*	✓*	✓	✓
`mw`	□	✗	✗	✗	✗	✗	✗	✗	✓‡*	✓	✓	✓
	△	✗	✗	✗	✓	✗	✗	✗	✓*	✓**	✓	✓
`lak-tag`	□	✗	✗	✗	✗	✗	✗	✗	✓*	✓	✓	✓
`feldhofer`	□	✗	✗	✗	✗	✗	✓	✗	✓*	✓	✓	✓
`euf-key-secrecy`		✗	✓‡	✗	✓‡	✗	✓‡	✓‡	✓‡	✓‡	✓‡	✓‡
`ddh`	□	✗	✗	✗	✗	✗	✓‡	✗	✗	✓‡	✓‡	✓‡
	△	✗	✗	✗	✗	✗	✓‡	✗	✗	✓‡	✓‡	✓‡

○ vanilla VAMPIRE (without Section 5 adjustments)
● using the added decision procedures
□ authentication
■ injective authentication
△ used as a lemma for unlinkability in SQUIRREL
‡ made use of lemmas (proven in CRYPTOVAMPIRE)
* split into two queries, one for each direction of the double implication
** the solver succeded on one side of the equivalence

Figure 5: CRYPTOVAMPIRE experiments

`feldhofer` [35], `mw` [36] and `ddh` [37] protocols (the □ in Fig. 5). We follow very closely [19]'s modelling of these protocols.

We note that [19] uses the examples we considered to further verify observational equivalence hyperproperties (e.g., unlikability or strong-secrecy in the case of these protocols) whose proofs rely on the initial trace properties. CRYPTOVAMPIRE does not yet support such hyperproperties. However, we show that CRYPTOVAMPIRE can verify some of the lemmas that are used to prove those properties in SQUIRREL [19] (marked with a △ in Fig. 5). Indeed, a proof of observational equivalence in the CCSA may require trace properties to prune out (nearly) impossible traces, or rewriting equal terms according to $\doteq$. Both kinds of properties are supported in CRYPTOVAMPIRE, which can therefore be used to facilitate proofs with SQUIRREL. Indeed, since both tools use the same underlying theory, results in CRYPTOVAMPIRE are immediately valid in SQUIRREL (and vice versa) (seeAppendix E).

Experimental Analysis. Our experimental results are summarized in Fig. 5.

We show the effectiveness of our preprocessing by gradually increasing its impact in Fig. 5. We start with only preprocessing the definition of the subterm relations (see Section 5.1). We continue with the instance preprocessing (see Section 5.3.1), which, combined with our saturation modifications (see Section 5.2), enables us to verify all considered protocols. However, in this mode, CRYPTOVAMPIRE often needs guidance to split double implications into two queries (indicated via ✓* in Fig. 5), or use further lemmas to prove (✓‡ in Fig. 5).

We also tested CRYPTOVAMPIRE using the incomplete simplification of the logic introduced in Section 5.3.2. In this mode, we observe that CRYPTOVAMPIRE overall gains in capabilities despite its incomplete strategy/heuristics. Fig. 5 also highlights the limits of this heuristic. Indeed, `euf-key-secrecy` and `ddh` required very simple auxiliary lemmas (hinting which axioms are to be instantiated with which messages) to trigger the right preprocessing. In the case of `ddh`, this lemma is not even passed on to the theorem prover as it is only used for the preprocessing but not for the proof.

We conducted a performance evaluation based on the previous benchmarks, using a machine with 64 cores and 128GB of RAM and running VAMPIRE, Z3, and CVC5 as a verification backend for CRYPTOVAMPIRE, measuring the time required by the fastest among them. CRYPTOVAMPIRE is capable of verifying the considered examples in a few dozen milliseconds.

In addition, CRYPTOVAMPIRE shows very competitive performance even against CRYPTOVERIF [14], the state-of-the-art automated cryptographic protocol verifier supporting computational security proofs. The results of our comparative performance evaluation are summarized in Fig. 6.

7. Conclusion and Further Work

We introduced CRYPTOVAMPIRE, the first fully automated cryptographic protocol verifier supporting the CCSA. A core contribution is the first-order encoding of CCSA along with dedicated proof techniques. We further designed tailored reasoning procedures and heuristics to enable leveraging state-of-the-art first-order theorem proving in CRYPTOVAMPIRE, as demonstrated by our experimental results.

As future work, we are interested in extending CRYPTOVAMPIRE to support compositional proof techniques [22]. Furthermore, we plan to integrate axioms to reason about

	CryptoVerif	CryptoVampire
`basic-hash`	52.8 ms	21.4 ms
`hash-lock`	62.6 ms	24.4 ms
`lak-tag`	61.9 ms	30.4 ms
`feldhofer`	60.9 ms	31.6 ms
`mw`	71.9 ms	38.3 ms

Figure 6: Runtimes of CryptoVampire and CryptoVerif for authentication queries

post-quantum security [21]. We also plan to integrate relational reasoning into CryptoVampire to support indistinguishability proofs [16].

Acknowledgments. This work was partially supported by the European Research Council (ERC) through the Consolidator Grants ARTIST 101002685 and Browsec 771527; the TU Wien Doctoral College SecInt; the Austrian Science Fund (FWF) through the SpyCoDe SFB projects F8504 and F8510-N; the Vienna Science and Technology Fund (WWTF) through [ForSmart Grant ID: 10.47379/ICT22007]; the Austrian Research Promotion Agency (FFG) through the the COMET K1 SBA and ABC; and the the Christian Doppler Research Association through the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

References

[1] G. Lowe, “An attack on the Needham-Schroeder public-key authentication protocol,” *Information Processing Letters*, vol. 56, no. 3, pp. 131–133, Nov. 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020019095001442

[2] N. J. Al Fardan and K. G. Paterson, “Lucky Thirteen: Breaking the TLS and DTLS Record Protocols,” in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 526–540.

[3] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, “Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability,” in *Proceedings 2019 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_09-4_Malavolta_paper.pdf

[4] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella-Béguelin, “Proving the TLS Handshake Secure (As It Is),” in *Advances in Cryptology – CRYPTO 2014*. Springer, Berlin, Heidelberg, 2014, pp. 235–255. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-44381-1_14

[5] B. Lipp, B. Blanchet, and K. Bhargavan, “A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, Jun. 2019, pp. 231–246.

[6] A. Koutsos, “The 5G-AKA Authentication Protocol Privacy,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, Jun. 2019, pp. 464–479.

[7] G. Lowe, “A hierarchy of authentication specifications,” in *Proceedings 10th Computer Security Foundations Workshop*, Jun. 1997, pp. 31–43.

[8] M. R. Clarkson and F. B. Schneider, “Hyperproperties,” in *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, USA, 23-25 June 2008*. IEEE Computer Society, 2008, pp. 51–65. [Online]. Available: https://doi.org/10.1109/CSF.2008.7

[9] D. Dolev and A. Yao, “On the security of public key protocols,” *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983.

[10] B. Blanchet, “An Efficient Cryptographic Protocol Verifier Based on Prolog Rules,” in *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, ser. CSFW ’01. USA: IEEE Computer Society, Jun. 2001, p. 82.

[11] S. Meier, B. Schmidt, C. Cremers, and D. Basin, “The TAMARIN Prover for the Symbolic Analysis of Security Protocols,” in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer, 2013, pp. 696–701.

[12] K. Bhargavan and G. Leurent, “Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH,” in *Proceedings 2016 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2016. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/transcript-collision-attacks-breaking-authentication-tls-ike-ssh.pdf

[13] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, “Computer-Aided Security Proofs for the Working Cryptographer,” in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed. Berlin, Heidelberg: Springer, 2011, pp. 71–90.

[14] B. Blanchet, “A computationally sound mechanized prover for security protocols,” in *2006 IEEE Symposium on Security and Privacy (S&P’06)*, May 2006, pp. 15 pp.–154.

[15] G. Bana and H. Comon-Lundh, “Towards Unconditional Soundness: Computationally Complete Symbolic Attacker,” in *Principles of Security and Trust*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, P. Degano, and J. D. Guttman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7215, pp. 189–208. [Online]. Available: http://link.springer.com/10.1007/978-3-642-28641-4_11

[16] ——, “A computationally complete symbolic attacker for equivalence properties,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 609–620.

[17] G. Bana, K. Hasebe, and M. Okada, “Computationally complete symbolic attacker and key exchange,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 1231–1246.

[18] D. Baelde, A. Koutsos, and J. Lallemand, “A Higher-Order Indistinguishability Logic for Cryptographic Reasoning,” in *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, Jun. 2023, pp. 1–13.

[19] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau, “An Interactive Prover for Protocol Verification in the Computational Model,” in *SP 2021 - 42nd IEEE Symposium on Security and Privacy*, San Fransisco / Virtual, United States, May 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03172119

[20] D. Baelde, S. Delaune, A. Koutsos, and S. Moreau, “Cracking the Stateful Nut,” in *CSF 2022 - 35th IEEE Computer Security Foundations Symposium*. Haifa, Israel: IRISA, Aug. 2022, Research Report. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03500056

[21] C. Cremers, C. Fontaine, and C. Jacomme, “A Logic and an Interactive Prover for the Computational Post-Quantum Security of Protocols,” in *2022 IEEE Symposium on Security and Privacy (SP)*, May 2022, pp. 125–141.

[22] H. Comon, C. Jacomme, and G. Scerri, "Oracle Simulation: A Technique for Protocol Composition with Long Term Shared Secrets," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. Virtual Event USA: ACM, Oct. 2020, pp. 1427–1444. [Online]. Available: https://dl.acm.org/doi/10.1145/3372297.3417229

[23] K. N. Venkataraman, "Decidability of the purely existential fragment of the theory of term algebras," *Journal of the ACM*, vol. 34, no. 2, pp. 492–510, Apr. 1987. [Online]. Available: https://dl.acm.org/doi/10.1145/23005.24037

[24] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds. Berlin, Heidelberg: Springer, 2008, pp. 337–340.

[25] L. Kovács and A. Voronkov, "First-Order Theorem Proving and Vampire," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer, 2013, pp. 1–35.

[26] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar, "Cvc5: A Versatile and Industrial-Strength SMT Solver," in *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, D. Fisman and G. Rosu, Eds., vol. 13243. Springer, 2022, pp. 415–442. [Online]. Available: https://doi.org/10.1007/978-3-030-99524-9_24

[27] M. Brusó, K. Chatzikokolakis, and J. den Hartog, "Formal Verification of Privacy for RFID Systems," in *2010 23rd IEEE Computer Security Foundations Symposium*, Jul. 2010, pp. 75–88.

[28] L. Kovács, S. Robillard, and A. Voronkov, "Coming to terms with quantified reasoning," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '17. New York, NY, USA: Association for Computing Machinery, Jan. 2017, pp. 260–270. [Online]. Available: https://dl.acm.org/doi/10.1145/3009837.3009887

[29] C. Cremers, C. Jacomme, and P. Lukert, "Subterm-based proof techniques for improving the automation and scope of security protocol analysis," in *CSF 2022 - 36th IEEE Computer Security Foundations Symposium*, Dubrovnik, Croatia, Aug. 2022, pp. 1–14. [Online]. Available: https://eprint.iacr.org/2022/1130

[30] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949. [Online]. Available: https://ieeexplore.ieee.org/document/6769090

[31] T. Skolem, "Logisch-kombinatorische untersuchungen über die erfüllbarkeit oder bewiesbarkeit mathematischer sätze nebst einem theorem über dichte mengen," in *Selected Works in Logic*, 1920.

[32] A. Bundy, "Chapter 13 - The Automation of Proof by Mathematical Induction," in *Handbook of Automated Reasoning*, ser. Handbook of Automated Reasoning, A. Robinson and A. Voronkov, Eds. Amsterdam: North-Holland, Jan. 2001, pp. 845–911. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444508133500151

[33] A. Juels and S. A. Weis, "Defining strong privacy for RFID," *ACM Transactions on Information and System Security*, vol. 13, no. 1, pp. 7:1–7:23, Nov. 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1609956.1609963

[34] L. Hirschi, D. Baelde, and S. Delaune, "A method for unbounded verification of privacy-type properties," *Journal of Computer Security*, vol. 27, no. 3, pp. 277–342, Jan. 2019. [Online]. Available: https://doi.org/10.3233/JCS-171070

[35] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, ser. Lecture Notes in Computer Science, M. Joye and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer, 2004, pp. 357–370.

[36] D. Molnar and D. Wagner, "Privacy and security in library RFID: Issues, practices, and architectures," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, ser. CCS '04. New York, NY, USA: Association for Computing Machinery, Oct. 2004, pp. 210–219. [Online]. Available: https://dl.acm.org/doi/10.1145/1030083.1030112

[37] "IT Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques," International Organization for Standardization, Geneva, CH, Standard, Jan. 2019.

[38] D. E. Knuth and P. B. Bendix, "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebra*, JOHN. Leech, Ed. Pergamon, Jan. 1970, pp. 263–297. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978008012975450028X

[39] J. Herbrand, *Recherches sur la théorie de la démonstration*, 1930. [Online]. Available: http://eudml.org/doc/192791

[40] C. Barrett, P. Fontaine, and C. Tinelli, "The SMT-LIB Standard: Version 2.6," Department of Computer Science, The University of Iowa, Tech. Rep., 2017.

[41] L. Bachmair, H. Ganzinger, D. McAllester, and C. Lynch, "Chapter 2 - Resolution Theorem Proving," in *Handbook of Automated Reasoning*, ser. Handbook of Automated Reasoning, A. Robinson and A. Voronkov, Eds. Amsterdam: North-Holland, Jan. 2001, pp. 19–99. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444508133500047

Appendix A. Complete semantics

We assume that the following functions are part of $\mathcal{F}$: IF _ THEN _ ELSE _, $\ll_{;}$ _ $\gg$, $\emptyset$, `true`, `false` and fail.

We use a pseudo-Dirac function in Fig. 7 for conciseness:

$$\delta(P) \coloneqq \begin{cases} \texttt{true} & \text{if } P \\ \texttt{false} & \text{otherwise} \end{cases}$$

Definition 10 (Previous Step). *Given* $\mathbb{T}$ *and a step* $a \in \mathbb{S} \cup \{\mathsf{undef}\}$*, we define the* previous step $\overline{\mathsf{pred}}_{\mathbb{T}}(a)$ *as such:*

$$\overline{\mathsf{pred}}_{\mathbb{T}}(a) \coloneqq \begin{cases} \mathsf{init} & \textit{if } a \textit{ is } \mathsf{init} \\ \mathsf{undef} & \textit{if } a \textit{ is } \mathsf{undef} \\ \max_{\leq_{\mathbb{S}}}\{b \mid b \in \mathbb{S}, b <_{\mathbb{S}} a\} & \textit{otherwise} \end{cases} \tag{39}$$

Definition 11 (Unfolding). *Given a trace* $\mathbb{T}$*, we unfold terms of* $\mathscr{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ *into* $\mathscr{BC}(\mathcal{N}_{\mathrm{bc}}, \mathcal{F}_{\mathrm{bc}}, \mathcal{G}_{\mathrm{bc}})$ *according to Fig. 7, with*

$$\mathcal{N}_{\mathrm{bc}} \coloneqq \left\{ \mathsf{n}_{\vec{\imath}} \middle| \mathsf{n}[\vec{\_}] \in \mathcal{N}, \vec{\imath} \in \mathbb{I}^{\|\vec{\_}\|} \right\} \tag{40}$$

$$\mathcal{F}_{\mathrm{bc}} \coloneqq \left\{ f_{\vec{\imath}} \middle| f[\vec{\_}] \in \mathcal{F}, \vec{\imath} \in \mathbb{I}^{\|\vec{\_}\|} \right\} \tag{41}$$

$$\mathcal{G}_{\mathrm{bc}} \coloneqq \{\mathsf{att}\} \tag{42}$$

Appendix B. Base Axioms

Notations. *In this section, we use* α *and* β *to refer to indices and/or timepoints when the distinction between the two is irrelevant.*

$$\underline{\mathbf{i}}_{\mathbb{T}} := \sigma_{\mathbb{I}}(\mathbf{i})$$
$$(\mathrm{pred}(T))^{\mathbb{T}} := \overline{\mathsf{pred}}_{\mathbb{T}}\Big((T)^{\mathbb{T}}\Big)$$
$$\big(\mathsf{a}[\vec{I}]\big)^{\mathbb{T}} := \begin{cases} \mathsf{a}\Big[[\vec{I}]^{\mathbb{T}}\Big] & \text{if it is in } \mathbb{S} \\ \mathsf{undef} & \text{otherwise} \end{cases}$$
$$[T < T']^{\mathbb{T}} := \delta\Big((T)^{\mathbb{T}} <_{\mathbb{S}} (T')^{\mathbb{T}}\Big)$$
$$[T = T']^{\mathbb{T}} := \delta\Big((T)^{\mathbb{T}} = (T')^{\mathbb{T}}\Big)$$
$$[I = I']^{\mathbb{T}} := \delta(\underline{I}_{\mathbb{T}} = \underline{I'}_{\mathbb{T}})$$
$$[\mathrm{happens}(T)]^{\mathbb{T}} := \delta\Big((T)^{\mathbb{T}} \neq \mathsf{undef}\Big)$$
$$\big[\mathsf{n}[\vec{I}]\big]^{\mathbb{T}} := \mathsf{n}_{\underline{\vec{I}}_{\mathbb{T}}}$$
$$\big[f[\vec{I}](\vec{t})\big]^{\mathbb{T}} := f_{\underline{\vec{I}}_{\mathbb{T}}}\Big([\vec{t}]^{\mathbb{T}}\Big)$$
$$[\mathrm{input}(T)]^{\mathbb{T}} := \mathsf{input}^{\mathbb{T}}_{(T)^{\mathbb{T}}}$$
$$\big[\mathsf{c}[\vec{I}]!(T)\big]^{\mathbb{T}} := \begin{cases} \emptyset & \text{if } (T)^{\mathbb{T}} \text{ is } \mathsf{undef} \\ \Big[u_{(T)^{\mathbb{T}}}\big(\mathsf{c}[\underline{\vec{I}}_{\mathbb{T}}]\big)\Big]^{\mathbb{T}} & \text{otherwise} \end{cases}$$

$$\mathsf{input}^{\mathbb{T}}_{\mathsf{init}}, \mathsf{input}^{\mathbb{T}}_{\mathsf{undef}}, \mathsf{frame}^{\mathbb{T}}_{\mathsf{undef}} := \emptyset \qquad \mathsf{exec}^{\mathbb{T}}_{\mathsf{undef}} := \mathtt{false}$$

$$\mathsf{exec}^{\mathbb{T}}_{\mathsf{init}} := \mathtt{true} \qquad \mathsf{frame}^{\mathbb{T}}_{\mathsf{init}} := \mathrm{msg}(\mathsf{init})$$

$$\mathsf{input}^{\mathbb{T}}_{a} := \mathsf{att}\Big(\mathsf{frame}^{\mathbb{T}}_{\overline{\mathsf{pred}_{\mathbb{T}}(a)}}\Big)$$

$$\mathsf{exec}^{\mathbb{T}}_{a} := \mathrm{cond}(a) \mathbin{\bar\wedge} \mathsf{exec}^{\mathbb{T}}_{\overline{\mathsf{pred}_{\mathbb{T}}(a)}}$$

$$\mathsf{frame}^{\mathbb{T}}_{a} := \ll \mathsf{exec}^{\mathbb{T}}_{a}; \ll \textsc{If } \mathsf{exec}^{\mathbb{T}}_{a} \textsc{ Then } \mathrm{msg}(a) \\ \textsc{Else } \mathsf{fail}; \mathsf{frame}^{\mathbb{T}}_{\overline{\mathsf{pred}_{\mathbb{T}}(a)}} \gg\gg$$

$$\textsc{Find } \vec{\imath} \textsc{ Such that } t_1[\vec{\imath}] \textsc{ Then } t_2[\vec{\imath}] \textsc{ Else } t_3 := \\ \textsc{If } t_1[\vec{\mathbf{i}}_1] \textsc{ Then } t_2[\vec{\mathbf{i}}_1] \textsc{ Else } \dots \\ \textsc{If } t_1[\vec{\mathbf{i}}_n] \textsc{ Then } t_2[\vec{\mathbf{i}}_n] \textsc{ Else } t_3$$

$$\textsc{Find } \vec{\tau} \textsc{ Such that } t_1[\vec{\tau}] \textsc{ Then } t_2[\vec{\tau}] \textsc{ Else } t_3 := \\ \textsc{If } t_1[\vec{\boldsymbol{\tau}}_1] \textsc{ Then } t_2[\vec{\boldsymbol{\tau}}_1] \textsc{ Else } \dots \\ \textsc{If } t_1[\vec{\boldsymbol{\tau}}_k] \textsc{ Then } t_2[\vec{\boldsymbol{\tau}}_k] \textsc{ Else } t_3$$

Where $\{\vec{\mathbf{i}}_1, \dots, \vec{\mathbf{i}}_n\} = \mathbb{I}^{\|\vec{\imath}\|}$ and $\{\vec{\boldsymbol{\tau}}_1, \dots, \vec{\boldsymbol{\tau}}_k\} = \mathcal{S}_{\mathbb{I}}^{\|\vec{\tau}\|}$.

Figure 7: Unfolding of $\mathbb{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$

$$|A \mathbin{\bar\wedge} B| \approx |A| \wedge |B| \quad (43) \qquad |\mathtt{true}| \approx \top \quad (46)$$
$$|A \mathbin{\bar\vee} B| \approx |A| \vee |B| \quad (44) \qquad |\mathtt{false}| \approx \bot \quad (47)$$
$$|\bar\neg A| \approx \neg |A| \quad (45) \qquad \big|\bar\exists \vec{\alpha}.\ t\big| \approx \exists \vec{\alpha}.\ |t| \quad (48)$$
$$\big|\bar\forall \vec{\alpha}.\ t\big| \approx \forall \vec{\alpha}.\ |t| \quad (49)$$
$$|A \mathbin{\bar\Rightarrow} B| \approx |A| \Rightarrow |B| \quad (50)$$

Where $A \approx B$ stands for $\mathcal{L}^\eta(A)(\rho) = \mathcal{L}^\eta(B)(\rho)$ for all η and ρ.

Figure 8: Collection of valid boolean axioms

Property 9. *In any model* If _ Then _ Else _, `true` *and* `false` *have the function evaluation that we would usually expect, the equations of Fig. 8 hold.*

Proof: Let $\mathcal{L}$ be any model in which If _ Then _ Else _, `true` and `false` have the function evaluation that we would usually expect. Then equations 43-47 and 50 hold using a truth table on $\Big[\!\Big[[A]^{\mathbb{T}}\Big]\!\Big]^{\mathbb{M}}_{\mathsf{bc}}$ and $\Big[\!\Big[[B]^{\mathbb{T}}\Big]\!\Big]^{\mathbb{M}}_{\mathsf{bc}}$.

(48) If $\mathcal{L}^\eta(\exists\alpha.\ |t|)(\rho) = 1$ with $\alpha = \mathrm{fv}(t)$, let $\boldsymbol{\alpha}_0$ be an element such that $\mathcal{L}^\eta(|t\{\alpha \mapsto \boldsymbol{\alpha}_0\}|)(\rho) = 1$ and $\boldsymbol{\beta}$ an element from $\mathbb{T}$ (i.e., an element of $\mathbb{I}$ or $\mathcal{S}_{\mathbb{I}}$) that unfolds like $\boldsymbol{\alpha}_0$ (there has to be one, as unfolding does not change member of $\mathbb{T}$). Then $\mathcal{L}^\eta(|t\{\alpha \mapsto \boldsymbol{\beta}\}|)(\rho) = 1$. We conclude by the construction of the symbolic existential quantifier (see (8)) and the unfolding of lookups. Else $\mathcal{L}^\eta(\exists\alpha.\ |t|)(\rho) = 0$ and the result is trivial.

(49) Consequence of (48).

□

This serves as a proof of Property 4.

Appendix C. Soundness Theorems

In this section, we formally define the notion of formulas with bounded Skolem normal form and proceed to prove Theorem 3.

Notations. *In this section, we use α and β to refer to indices and/or timepoints when the distinction between the two is irrelevant.*

We write $\vDash \phi$ to say that ϕ is valid in FOL, i.e., ϕ is satisfied by any model.

C.1. Some definitions

Let us begin by introducing some definitions. For conciseness, let $\mathcal{E}_0 := \mathscr{E}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$. We define $\mathcal{S}\mathrm{k}$ as a set of Skolem symbols, and $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k})$ as the logic extending $\mathcal{E}_0$ with those Skolem symbols.

Definition 12 (Sequences of Sets of Random Tapes)**.** *We write $\Omega^{\mathbb{N}}$ as the set of sequences of Ω. We also define the ring $(\Omega, \Delta, \cap)$ as follows:*

1) $(A_\eta)_{\eta\in\mathbb{N}} \mathbin{\Delta} (B_\eta)_{\eta\in\mathbb{N}} := (A_\eta \mathbin{\Delta} B_\eta)_{\eta\in\mathbb{N}}$;
2) $(A_\eta)_{\eta\in\mathbb{N}} \cap (B_\eta)_{\eta\in\mathbb{N}} := (A_\eta \cap B_\eta)_{\eta\in\mathbb{N}}$;

where Δ is the symmetric difference. We also include the union $\cup$ as $A \cup B \coloneqq A \,\Delta\, B \,\Delta\, (A \cap B)$.

Definition 13 (Negligible Family)**.** *A family (or sequence) $(\mathfrak{S}_\eta)_{\eta\in\mathbb{N}}$ of subsets of Ω is* negligible *if $\mathbb{P}\mathrm{rob}_\rho(\mathfrak{S}_\eta) = \mathrm{negl}(\eta)$. We write Ω_{negl} the set of such sequences.*

Definition 14 (Skolem Model)**.** *A Skolem model M_{sk} maps terms of $\mathscr{T}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k})$ to $\mathcal{E}_0$ such that it is the identity on every symbol except those of $\mathcal{S}\mathrm{k}$.*

Definition 15 (Finite Term)**.** *A* domain function *$\mathscr{D}$ over $\mathcal{S}\mathrm{k}$ is a function from $\mathscr{T}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k})$ to the finite sets of $\mathcal{E}_0$ such that for all $\boldsymbol{u} = \mathfrak{f}(\vec{\boldsymbol{v}})$ that is not of the form $\mathrm{sk}(\_)$ (with $\mathrm{sk} \in \mathcal{S}\mathrm{k}$), we have $\mathscr{D}(\mathfrak{f}(\vec{\boldsymbol{v}})) = \left\{\mathfrak{f}(\vec{\boldsymbol{u}}) \middle| \vec{\boldsymbol{u}} \in \mathscr{D}(\vec{\boldsymbol{v}})\right\}$ where $\mathfrak{f}$ stands for any application-like construction (i.e., anything but constant symbols).*

Definition 16 (Extended Cryptographic Model)**.** *An* extended cryptographic model *$\mathfrak{K}$ is composed of*

- *a cryptographic model $\mathcal{L}$;*
- *a family of Skolem models $\left(\mathsf{M}_{\mathrm{sk}}^{\eta,\rho}\right)_{\eta,\rho}$;*
- *a domain function $\mathscr{D}$ over $\mathcal{S}\mathrm{k}$.*

such that

1) *$\mathfrak{K}^\eta(|t|)(\rho) = \mathcal{L}^\eta\left(\left|\mathsf{M}_{\mathrm{sk}}^{\eta,\rho}(t)\right|\right)(\rho)$*
2) *$\mathfrak{K}^\eta(S)(\rho) = \mathcal{L}^\eta\left(\mathsf{M}_{\mathrm{sk}}^{\eta,\rho}(S)\right)$*
3) *for all term u, we have*

$$\mathfrak{K}^\eta\left(\bigvee_{v\in\mathscr{D}(u),\, \vec{\beta}=\mathrm{fv}(v)} \exists\vec{\beta}.\ v = u\right)(\rho) = 1 \tag{51}$$

where $\vec{\beta}$ is a list of variables over timepoints and/or indices.

Definition 17 (Model Extension)**.** *Let L be a model over a logic $\mathcal{E}$. A model K over $\mathcal{E}'$* extends *L when $\mathcal{E} \subseteq \mathcal{E}'$ and for all η, ρ, and $\varphi \in \mathcal{E}$, we have $L^\eta(\varphi)(\rho) = K^\eta(\varphi)(\rho)$. We write it $L \trianglelefteq K$. We also say that K* covers *L.*

Definition 18 (Formula with Bounded Skolem Normal Form)**.** *A formula $\varphi \in \mathcal{E}_0$ with $\vec{u} = \mathrm{fv}(\varphi)$ has a* bounded Skolem normal form *if we can find a finite $\mathcal{S}\mathrm{k}$ and a domain function $\mathscr{D}$ over $\mathcal{S}\mathrm{k}$ such that:*

1) *the Skolem normal form $\forall\vec{u},\vec{v}.\ \varphi_{\mathrm{sk}}$ of $\forall\vec{u}.\ \varphi$ uses only symbols of $\mathscr{T}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k})$;*
2) *for all cryptographic model $\mathcal{L}$, we can find an extended cryptographic model $\mathfrak{K}^{\mathcal{L}}$ using $\mathscr{D}$ such that:*
   a) *$\mathcal{L} \trianglelefteq \mathfrak{K}^{\mathcal{L}}$;*
   b) *for all $\vec{\boldsymbol{u}} \in \mathcal{E}_0$,*

$$\begin{aligned} &\text{if } \mathcal{L}^\eta(\varphi\{\vec{u} \mapsto \vec{\boldsymbol{u}}\})(\rho) = 1 \\ &\qquad \text{then } \mathfrak{K}^\eta(\forall\vec{v}.\ \varphi_{\mathrm{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}\})(\rho) = 1 \end{aligned} \tag{52}$$

C.2. Some properties

Property 10 (Model Extension as Order)**.** *$\trianglelefteq$ is a partial order over models and a preorder over sets of models.*

Property 11. *Ω_{negl} is an ideal of $\left(\Omega^{\mathbb{N}}, \Delta, \cap\right)$.*

Proof:

1) Let $(A_\eta)_{\eta\in\mathbb{N}}, (B_\eta)_{\eta\in\mathbb{N}} \in \Omega_{\mathrm{negl}}$:

$$\begin{aligned} &\mathbb{P}\mathrm{rob}_\rho(A_\eta \,\Delta\, B_\eta) \\ &\quad = \mathbb{P}\mathrm{rob}_\rho(A_\eta) + \mathbb{P}\mathrm{rob}_\rho(B_\eta) - 2\mathbb{P}\mathrm{rob}_\rho(A_\eta \cap B_\eta) \\ &\qquad \le \mathbb{P}\mathrm{rob}_\rho(A_\eta) + \mathbb{P}\mathrm{rob}_\rho(B_\eta) = \mathrm{negl}(\eta) \end{aligned}$$

2) Let $(A_\eta)_{\eta\in\mathbb{N}} \in \Omega^{\mathbb{N}}$ and $(B_\eta)_{\eta\in\mathbb{N}} \in \Omega_{\mathrm{negl}}$:

$$\mathbb{P}\mathrm{rob}_\rho(A_\eta \cap B_\eta) \le \mathbb{P}\mathrm{rob}_\rho(B_\eta) = \mathrm{negl}(\eta)$$

□

$$\frac{C \vDash_{\mathbb{P}} A \qquad C \vDash_{\mathbb{P}} B}{C \vDash_{\mathbb{P}} A \wedge B}\wedge\text{-INT} \qquad \frac{C \vDash_{\mathbb{P}} A \wedge B}{C \vDash_{\mathbb{P}} A}\wedge\text{-EX-1}$$

$$\frac{C \vDash_{\mathbb{P}} A \wedge B}{C \vDash_{\mathbb{P}} B}\wedge\text{-EX-2} \qquad \frac{C \vDash_{\mathbb{P}} A}{C \vDash_{\mathbb{P}} A \vee B}\vee\text{-INT-1}$$

$$\frac{C \vDash_{\mathbb{P}} B}{C \vDash_{\mathbb{P}} A \vee B}\vee\text{-INT-2} \qquad \frac{C \vDash_{\mathbb{P}} A \qquad \vDash A \Rightarrow B}{C \vDash_{\mathbb{P}} B}\text{MP}$$

$$\frac{C' \vDash_{\mathbb{P}} A \qquad C \trianglelefteq C'}{C \vDash_{\mathbb{P}} A}\text{RSTR} \qquad \frac{\vDash A}{C \vDash_{\mathbb{P}} A}\text{FOL}$$

Note that in MP, $A \Rightarrow B$ needs to be valid in *FOL*.

Figure 9: Some Sequent Rules

Property 12. *The rules in Fig. 9 are sound.*

Proof: Consequences of Property 11. □

Lemma 1. *If Δ and Γ have a bounded Skolem normal form, then we can find a $\mathcal{S}\mathrm{k}$ and $\mathscr{D}$ such that both formulas' Skolem normal forms are in $\mathscr{T}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k})$ with $\mathscr{D}$, and we can make use of the same $\mathfrak{K}$.*

Proof: Δ has a bounded Skolem normal form with $\mathcal{S}\mathrm{k}_\Delta$ and $\mathscr{D}_\Delta$, and Γ with $\mathcal{S}\mathrm{k}_\Gamma$ and $\mathscr{D}_\Gamma$. W.l.o.g. we can assume $\mathcal{S}\mathrm{k}_\Delta \cap \mathcal{S}\mathrm{k}_\Gamma = \varnothing$. Let $\mathcal{S}\mathrm{k} = \mathcal{S}\mathrm{k}_\Delta \cup \mathcal{S}\mathrm{k}_\Gamma$, $\vec{u} = \mathrm{fv}(\Delta)$, and $\vec{v} = \mathrm{fv}(\Gamma)$.

We define:

$$\mathscr{D}(\boldsymbol{u}) = \begin{cases} \bigcup_{\vec{\boldsymbol{v}}\in\mathscr{D}(\vec{\boldsymbol{u}})} \mathscr{D}_\Delta(\mathrm{sk}_\Delta(\vec{\boldsymbol{v}})) \\ \qquad\qquad \text{if } \boldsymbol{u} = \mathrm{sk}_\Delta(\vec{\boldsymbol{u}}) \text{ with } \mathrm{sk} \in \mathcal{S}\mathrm{k}_\Delta \\ \bigcup_{\vec{\boldsymbol{v}}\in\mathscr{D}(\vec{\boldsymbol{u}})} \mathscr{D}_\Gamma(\mathrm{sk}_\Gamma(\vec{\boldsymbol{v}})) \\ \qquad\qquad \text{if } \boldsymbol{u} = \mathrm{sk}_\Gamma(\vec{\boldsymbol{u}}) \text{ with } \mathrm{sk} \in \mathcal{S}\mathrm{k}_\Gamma \\ \left\{\mathfrak{f}(\vec{\boldsymbol{u}}) \middle| \vec{\boldsymbol{u}} \in \mathscr{D}(\vec{\boldsymbol{v}})\right\} \text{ if } \boldsymbol{u} = \mathfrak{f}(\vec{\boldsymbol{v}}) \end{cases} \tag{53}$$

Let us show that Δ has a bounded Skolem normal form with $\mathcal{S}\mathrm{k}_\Delta \cup \mathcal{S}\mathrm{k}_\Gamma$ and $\mathscr{D}$:

1) because $\mathscr{T}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k}_\Delta) \subseteq \mathscr{T}_{\mathrm{sk}}(\mathcal{E}_0, \mathcal{S}\mathrm{k}_\Delta \cup \mathcal{S}\mathrm{k}_\Gamma)$
2) Let $\mathcal{L}$ be a cryptographic model and $\mathfrak{K}_\Delta$ and $\mathfrak{K}_\Gamma$ its extension according to Definition 18's (2). Let us build $\mathfrak{K}$ such that $\mathfrak{K}_\Delta \trianglelefteq \mathfrak{K}$ and $\mathfrak{K}_\Gamma \trianglelefteq \mathfrak{K}$. We choose $\mathsf{M}_{\mathrm{sk}}^{\mathfrak{K}}\big|_{\mathcal{S}\mathrm{k}_\Delta} = \mathsf{M}_{\mathrm{sk}}^{\mathfrak{K}_\Delta}\Big|_{\mathcal{S}\mathrm{k}_\Delta}$ and $\mathsf{M}_{\mathrm{sk}}^{\mathfrak{K}}\big|_{\mathcal{S}\mathrm{k}_\Gamma} = \mathsf{M}_{\mathrm{sk}}^{\mathfrak{K}_\Gamma}\Big|_{\mathcal{S}\mathrm{k}_\Gamma}$. $\mathfrak{K}$ is fully defined.

Moreover, by induction, we show that $\mathfrak{K}$ follows Definition 16's (3) and Definition 17. Thus,

$$\mathfrak{K}_\Delta \trianglelefteq \mathfrak{K} \text{ and } \mathfrak{K}_\Gamma \trianglelefteq \mathfrak{K} \tag{54}$$

By Property 10 we get Definition 18's (2)a, and (54) gives us Definition 18's (2)b.

By symmetry, we also get the result for Γ. □

Property (7)**.** *The notion of bounded Skolem normal form is stable by conjunction and disjunction.*

Proof: Let Δ and Γ be two formulas with a bounded Skolem normal form using $\mathcal{S}\mathrm{k}$ and $\mathscr{D}$ (as per Lemma 1) and $\vec{u}$ and $\vec{v}$ be their respective free variables. Let $\star$ stand for $\vee$ or $\wedge$.

Let $\forall\vec{u},\vec{u}'.\ \Delta_{\mathrm{sk}}$ and $\forall\vec{v},\vec{v}'.\ \Gamma_{\mathrm{sk}}$ be the Skolem normal form of Δ and Γ respectively as defined in Definition 18's (1).

1) $\forall\vec{u},\vec{u}',\vec{v},\vec{v}'.\Delta_{\mathrm{sk}}\star\Gamma_{\mathrm{sk}}$ is a Skolem normal form of $\forall\vec{u},\vec{v}.\Delta\star\Gamma$ and is in $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k})$.
2) Let $\mathcal{L}$ be a cryptographic model and $\mathfrak{K}$ its extension as defined in Definition 18.
   a) $\mathcal{L} \trianglelefteq \mathfrak{K}$ by definition.
   b) For all $\vec{\boldsymbol{u}}$ and $\vec{\boldsymbol{v}}$, (55) is valid in FOL.

$$\begin{pmatrix}(\Delta\{\vec{u}\mapsto\vec{\boldsymbol{u}}\}\Rightarrow\forall\vec{u}'.\ \Delta_{\mathrm{sk}}\{\vec{u}\mapsto\vec{\boldsymbol{u}}\})\\ \wedge(\Gamma\{\vec{v}\mapsto\vec{\boldsymbol{v}}\}\Rightarrow\forall\vec{v}'.\ \Gamma_{\mathrm{sk}}\{\vec{v}\mapsto\vec{\boldsymbol{v}}\})\end{pmatrix}\Rightarrow\begin{pmatrix}(\Delta\star\Gamma)\{\vec{u},\vec{v}\mapsto\vec{\boldsymbol{u}},\vec{\boldsymbol{v}}\}\\ \Rightarrow\forall\vec{u}',\vec{v}'.\ (\Delta_{\mathrm{sk}}\star\Gamma_{\mathrm{sk}})\{\vec{u},\vec{v}\mapsto\vec{\boldsymbol{u}},\vec{\boldsymbol{v}}\}\end{pmatrix} \tag{55}$$

We conclude using MP and the definition of $\trianglelefteq$.

□

Before proving Property 8, we first prove this convenient property:

Property 13 (Model Extension)**.** *If $\mathcal{L}$ and $\mathfrak{K}$ agree on all quantifier-free terms of $\mathcal{E}_0$, then $\mathcal{L}\trianglelefteq\mathfrak{K}$.*

Proof: By induction over φ.

1) The base cases are trivial (they are quantifier-free formulas).

2) $\varphi\vee\varphi'$ and $\neg\varphi$ are also trivial due to the semantics of $\mathcal{L}$ and $\mathfrak{K}$.

3) If φ is of the form $\exists u.\ \varphi'$, we then suppose that the property holds for $\varphi'\{u\mapsto\boldsymbol{u}\}$ when $\boldsymbol{u}\in\mathcal{E}_0$ ($\boldsymbol{u}$ is quantifier free).

If $\mathcal{L}^\eta(\varphi)(\rho)=1$ then $\mathfrak{K}^\eta(\varphi)(\rho)=1$ as a term of $\mathcal{E}_0$ is also a term of $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k})$. If $\mathcal{L}^\eta(\varphi)(\rho)=0$ then let us assume that $\mathfrak{K}^\eta(\varphi)(\rho)=1$. It means that there is a term $\boldsymbol{u}$ of $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k})$ such that $\mathfrak{K}^\eta(\varphi'[\boldsymbol{u}])(\rho)=1$.

By Definition 16's (3) and superposition we can find a $\boldsymbol{v}\in\mathscr{D}(\boldsymbol{u})$ such that $\mathfrak{K}^\eta(\exists\vec{\alpha}.\ \varphi'\{u\mapsto\boldsymbol{v}\})(\rho)=1$ and $\vec{\alpha}=\mathrm{fv}(\boldsymbol{v})$. Thus, we can find a $\vec{\boldsymbol{\alpha}}$ such that $\mathfrak{K}^\eta(\varphi'\{u\mapsto\boldsymbol{v}_{\vec{\boldsymbol{\alpha}}}\})(\rho)=1$ where $\boldsymbol{v}_{\vec{\boldsymbol{\alpha}}}:=\boldsymbol{v}\{\vec{\alpha}\mapsto\vec{\boldsymbol{\alpha}}\}$.

Then let $\vec{\boldsymbol{\beta}}=\mathsf{M}_{\mathrm{sk}}(\vec{\boldsymbol{\alpha}})$, we still have $\mathfrak{K}^\eta\left(\varphi'\left\{u\mapsto\boldsymbol{v}_{\vec{\boldsymbol{\beta}}}\right\}\right)(\rho)=1$ with $\varphi'\left\{u\mapsto\boldsymbol{v}_{\vec{\boldsymbol{\beta}}}\right\}\in\mathcal{E}_0$ and $\boldsymbol{v}_{\vec{\boldsymbol{\beta}}}:=\boldsymbol{v}\left\{\vec{\alpha}\mapsto\vec{\boldsymbol{\beta}}\right\}$. This contradicts the induction hypothesis.

4) φ is a universal quantifier. The result is a consequence of 2 and 3.

□

Property (8)**.** *If φ is bounding and has a bounded Skolem normal form, then $\exists x.\ \varphi$ has a bounded Skolem normal form.*

Proof: Let $x\uplus\vec{u}=\mathrm{fv}(\varphi)$ and $\mathscr{D}_0$ and $\mathcal{S}\mathrm{k}_0$ be what $\forall x,\vec{u}.\ \varphi$ has a bounded Skolem normal form with. Let $\mathcal{L}$ be a cryptographic model and $\mathfrak{K}_0=\left(\mathsf{M}_{\mathrm{sk}}{}^{(0)},\dots\right)$ its extension as defined in Definition 18 for φ.

1) We define $\mathcal{S}\mathrm{k}:=\mathcal{S}\mathrm{k}_0\cup\{\mathrm{sk}(\_{}_1,\dots,\_{}_n)\}$ such that $\mathrm{sk}\notin\mathcal{S}\mathrm{k}$ and $\|\vec{u}\|=n$. Then $\forall\vec{u}.\ \exists x.\ \varphi$ can be skolemed in $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k})$. Indeed, let $\forall\vec{v}.\ \varphi_{\mathrm{sk}}$ be a Skolem normal form of $\forall x,\vec{u}.\ \varphi$, then $\forall\vec{u}.\varphi\{x\mapsto\mathrm{sk}(\vec{u})\}$ is a Skolem normal form of $\forall\vec{u}.\ \exists x.\ \varphi$.

2) Let us build $\mathfrak{K}$, an extension of $\mathcal{L}$ as in Definition 18's (2). We define:

$$\mathscr{D}\big(\mathrm{sk}'(\vec{u})\big)=\begin{cases}D_\varphi^{\vec{\boldsymbol{u}}}\cup\{\mathsf{fail}\} & \text{if } \mathrm{sk}'=\mathrm{sk}\\ \bigcup_{\vec{\boldsymbol{v}}\in\mathscr{D}(\vec{\boldsymbol{u}})}\mathscr{D}\big(\mathrm{sk}'(\vec{\boldsymbol{v}})\big) & \text{otherwise}\end{cases} \tag{56}$$

the rest of $\mathscr{D}$ is defined recursively by Definition 15. Notice that $\mathscr{D}$ is $\mathscr{D}_0$ over $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k}_0)$.

Since $\mathcal{L}^\eta(\_)(\rho)$ is a first-order model we can construct the set:

$$SK_{\vec{\boldsymbol{u}}}^{\eta,\rho}:=\begin{cases}\{t'|\mathcal{L}^\eta(\varphi\{x,\vec{u}\mapsto t',\vec{\boldsymbol{u}}\})(\rho)=1\}\\ \qquad\text{if } \mathcal{L}^\eta(\exists x.\ \varphi\{\vec{u}\mapsto\vec{\boldsymbol{u}}\})(\rho)=1\\ \{\mathsf{fail}\}\text{ otherwise}\end{cases} \tag{57}$$

By noticing that we can find an ordering $\prec_{\mathscr{E}}$ over the terms such that it has a smallest element (e.g., a KBO [38]), we build

$$\mathsf{M}_{\mathrm{sk}}^{\eta,\rho}(\phi):=\begin{cases}\min_{\prec_{\mathscr{E}}}\left(SK^{\eta,\rho}_{\mathsf{M}_{\mathrm{sk}}^{\eta,\rho}(\vec{\boldsymbol{u}})}\right) & \text{if } \phi=\mathrm{sk}(\vec{\boldsymbol{u}})\\ \mathsf{M}_{\mathrm{sk}}^{\eta,\rho(0)}(\phi) & \text{otherwise}\end{cases}$$

notice that M_{sk} is well defined as the number of sk strictly decreases at each recursive call. M_{sk} is a Skolem model that matches $\mathsf{M}_{\mathrm{sk}}{}^{(0)}$ over $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k}_0)$.

Let $\mathfrak{K}:=\big(\mathscr{D},\mathsf{M}_{\mathrm{sk}},\dots\mathfrak{K}_0\big)$. We note that $\mathfrak{K}$ is indeed an extended model: it verifies Definition 16's (3) because φ is bounding. Indeed, reusing Definition 8's notations, we have:

$$SK_{\vec{\boldsymbol{u}}}^{\eta,\rho}\subseteq D_\varphi^{\vec{\boldsymbol{u}}}\cup\{\mathsf{fail}\}\text{ (up to }\alpha\text{-renaming)} \tag{58}$$

Then

a) It agrees with $\mathfrak{K}_0$ all ground terms on $\mathscr{E}_{\mathrm{sk}}(\mathcal{E}_0,\mathcal{S}\mathrm{k}_0)$ (all its components are the same on over that set), thus it agrees with $\mathcal{L}$ on all ground terms of $\mathcal{E}_0$. Then by Property 13 we have $\mathcal{L}\trianglelefteq\mathfrak{K}$;
b) let $\vec{\boldsymbol{u}}\in\mathcal{E}_0$, let us assume we have $\mathcal{L}^\eta(\exists x.\ \varphi\{\vec{u}\mapsto\vec{\boldsymbol{u}}\})(\rho)=1$. By construction, we can find $t\in SK_{\vec{\boldsymbol{u}}}^{\eta,\rho}$ such that $\mathcal{L}^\eta(\varphi\{x,\vec{u}\mapsto t,\vec{\boldsymbol{u}}\})(\rho)=1$. Also,

by construction, we can suppose $\mathsf{M}_{\text{sk}}^{\eta,\rho}(\text{sk}(\vec{\boldsymbol{u}})) = \mathsf{M}_{\text{sk}}^{\eta,\rho}(t)$. Which lets us conclude.

□

C.3. The Theorems

This section provides a proof of Theorem 3.

Theorem (3)**.** *Let φ be a formula s.t. $\text{fv}(\varphi) = \vec{u}$, t a term, and $\mathbb{C}$ a set of cryptographic models such that*

1) *φ has a bounded Skolem normal form;*
2) *for all $\vec{\boldsymbol{v}}$, $\mathbb{C} \vDash_{\mathbb{P}} \varphi\{\vec{u} \mapsto \vec{\boldsymbol{v}}\}$;*
3) *$(\forall \vec{u}.\ \varphi) \Rightarrow |t|$ is valid in FOL;*

then $\mathbb{C} \vDash_{\mathbb{P}} |t|$.

Example (16)**.** *Recall Example 16. $\mathcal{E}_0$ rejects* (24)*.*

$$\forall \vec{\imath}, x.\ |\mathsf{n}[\vec{\imath}]| = |x| \Rightarrow \mathsf{n}[\vec{\imath}] \sqsubseteq x \tag{24}$$

At first glance, it seems we can recover Example 16 using Herbrand's Theorem [39]. We indeed can show the following lemma:

Theorem 5. *Let φ be a formula of $\mathcal{E}_0$ with $\vec{u} := \text{fv}(\varphi)$, t be a term of $\mathcal{E}_0$ and $\mathbb{C}$ a class of cryptographic models, if we can find $\mathbb{C}'$ and $\mathcal{S}\text{k}$ such that*

1) *$\forall \vec{u}, \vec{v}.\ \varphi_{\text{sk}}$ is a Skolem normal form of $\forall \vec{u}.\ \varphi$ and is part of $\mathscr{E}_{\text{sk}}(\mathcal{E}_0, \mathcal{S}\text{k})$;*
2) *$\mathbb{C} \trianglelefteq \mathbb{C}'$;*
3) *for all $\vec{\boldsymbol{u}} \in \mathscr{E}_{\text{sk}}(\mathcal{E}_0, \mathcal{S}\text{k})$, $\mathbb{C}' \vDash_{\mathbb{P}} \forall \vec{v}.\ \varphi_{\text{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}\}$;*
4) *$\vDash (\forall \vec{u}.\ \varphi) \Rightarrow |t|$ (in FOL);*

then $\mathbb{C} \vDash_{\mathbb{P}} |t|$.

Proof: (1) and (4) give us that $(\forall \vec{u}, \vec{v}.\ \varphi_{\text{sk}}) \wedge \neg|t|$ is unsatisfiable.

Then Herbrand's theorem [39] gives us $\vec{\boldsymbol{u}}_1, \ldots, \vec{\boldsymbol{u}}_n$, $\vec{\boldsymbol{v}}_1, \ldots, \vec{\boldsymbol{v}}_n$ such that $\left(\bigwedge_{k=1}^n \varphi_{\text{sk}}\{\vec{u}, \vec{v} \mapsto \vec{\boldsymbol{u}}_k, \vec{\boldsymbol{v}}_k\}\right) \wedge \neg|t|$ is unsatisfiable. Thus

$$\vDash \left(\bigwedge_{k=1}^{n} \forall \vec{v}.\ \varphi_{\text{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}_k\}\right) \Rightarrow |t|$$

We conclude using (2), (3), and Property 12. □

Unfortunately, item (3) is tricky to show as we have very little control over the interpretation of the $\boldsymbol{u}$ if it contains Skolems. We use the class of models from Definition 16 and formula with bounded Skolem normal to regain control:

Theorem 6. *Reusing the notation of Definition 18. Let $\varphi \in \mathcal{E}_0$ be a formula with a bound Skolem normal form and $\mathcal{L}$ a cryptographic models with $\vec{u} := \text{fv}(\varphi)$ such that for all ground $\vec{\boldsymbol{u}} \in \mathcal{E}_0$ we have $\mathcal{L} \vDash_{\mathbb{P}} \varphi\{\vec{u} \mapsto \vec{\boldsymbol{u}}\}$. Then for all $\vec{\boldsymbol{u}}' \in \mathscr{E}_{\text{sk}}(\mathcal{E}_0, \mathcal{S}\text{k})$ we have $\mathfrak{K}^{\mathcal{L}} \vDash_{\mathbb{P}} \forall \vec{v}. \varphi_{\text{sk}}\left\{\vec{u} \mapsto \vec{\boldsymbol{u}}'\right\}$.*

Lemma 2. *Let $\mathfrak{K}$ be as in Definition 16,*

$$\mathfrak{K}^\eta\left(\bigwedge\nolimits_{\alpha \in \mathbb{T}} \varphi\right)(\rho) = \mathfrak{K}^\eta(\forall \alpha.\ \varphi)(\rho) \tag{59}$$

$$\mathfrak{K}^\eta\left(\bigvee\nolimits_{\alpha \in \mathbb{T}} \varphi\right)(\rho) = \mathfrak{K}^\eta(\exists \alpha.\ \varphi)(\rho) \tag{60}$$

where α is an index or a timepoint. This naturally extends to quantification over multiple variables.

Proof: Let φ be a formula and α its free variable.

(59) If $\mathfrak{K}^\eta(\forall \alpha.\ \varphi[\alpha])(\rho) = 0$, then we can find $\boldsymbol{\alpha}$ such that $\mathfrak{K}^\eta(\varphi\{\alpha \mapsto \boldsymbol{\alpha}\})(\rho) = 0$. However, the unfolding $\boldsymbol{\beta}$ of $\mathsf{M}_{\text{sk}}^{\eta,\rho}(\boldsymbol{\alpha})$ is in $\mathbb{T}$. Thus, by construction of the unfolding and $\mathfrak{K}$, we get $\mathfrak{K}^\eta(\varphi\{\alpha \mapsto \boldsymbol{\beta}\})(\rho) = 0$. Hence, the equality.
Else $\mathfrak{K}^\eta(\forall \alpha.\ \varphi)(\rho) = 1$ and the result is trivial.

(60) Using (59).

□

Proof of Theorem 6: Let $\vec{\boldsymbol{u}}' \in \mathscr{E}_{\text{sk}}(\mathcal{E}_0, \mathcal{S}\text{k})$ and $\vec{\boldsymbol{u}} \in \mathscr{D}(\vec{\boldsymbol{u}}')$ with $\vec{\alpha} := \text{fv}(\vec{\boldsymbol{u}})$. We also use notations

$$\vec{\boldsymbol{u}}_{\vec{\boldsymbol{\beta}}} := \vec{\boldsymbol{u}}\left\{\vec{\alpha} \mapsto \vec{\boldsymbol{\beta}}\right\} \tag{61}$$

$$\Pi_{\vec{\boldsymbol{u}}} := \bigwedge\nolimits_{\vec{\boldsymbol{\alpha}} \in \mathbb{T}} \forall \vec{v}.\ \varphi_{\text{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}_{\vec{\boldsymbol{\alpha}}}\} \tag{62}$$

$$\Gamma := \bigwedge\nolimits_{\substack{\vec{\boldsymbol{u}} \in \mathscr{D}(\vec{\boldsymbol{u}}') \\ \vec{\alpha} = \text{fv}(\vec{\boldsymbol{u}})}} \forall \vec{\alpha}, \vec{v}.\ \varphi_{\text{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}\} \tag{63}$$

$$\Delta := \bigvee\nolimits_{\substack{\vec{\boldsymbol{v}} \in \mathscr{D}(\vec{\boldsymbol{u}}') \\ \vec{\alpha} = \text{fv}(\vec{\boldsymbol{v}})}} \exists \vec{\alpha}.\ \vec{\boldsymbol{v}} = \vec{\boldsymbol{u}}' \tag{64}$$

By assumption, for all $\vec{\boldsymbol{\alpha}} \in \mathcal{E}_0$ we have $\mathcal{L} \vDash_{\mathbb{P}} \varphi\{\vec{u} \mapsto \vec{\boldsymbol{u}}_{\vec{\boldsymbol{\alpha}}}\}$. Thus, by Definition 18's (2)b, $\mathfrak{K}^{\mathcal{L}} \vDash_{\mathbb{P}} \forall \vec{v}.\ \varphi_{\text{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}_{\vec{\boldsymbol{\alpha}}}\}$.

Then, rule $\wedge$-INT, gives us $\mathfrak{K}^{\mathcal{L}} \vDash_{\mathbb{P}} \Pi_{\vec{\boldsymbol{u}}}$. Finally, using Lemma 2, we get $\mathfrak{K}^{\mathcal{L}} \vDash_{\mathbb{P}} \forall \vec{\alpha}, \vec{v}.\ \varphi_{\text{sk}}\{\vec{u} \mapsto \vec{\boldsymbol{u}}_{\vec{\boldsymbol{\alpha}}}\}$.

Then, rule $\wedge$-INT, gives us $\mathfrak{K}^{\mathcal{L}} \vDash_{\mathbb{P}} \Gamma$. And we also have by construction of $\mathfrak{K}^{\mathcal{L}}$ (Definition 16's (3)) that $\mathfrak{K}^{\mathcal{L}} \vDash_{\mathbb{P}} \Delta$.

Moreover $\vDash \Gamma \wedge \Delta \Rightarrow \forall \vec{v}. \varphi_{\text{sk}}\left\{\vec{u} \mapsto \vec{\boldsymbol{u}}'\right\}$. Thus we can conclude using MP and $\wedge$-INT. □

Then Theorem 3 is the result of chaining Theorem 6 and Theorem 5.

Appendix D. Subterm definition

We define the overapproximated sets of subterms $\mathsf{st}_{\mathcal{P}}(t)$ for a protocol $\mathcal{P}$ used in Definition 9 in Fig. 10.

Let $\mathfrak{F}$ be a set of functions "ignored" by $\mathsf{st}_{\mathcal{P}}(t)$ and $\mathfrak{H}$ the set of terms using function as their head.

$$\mathfrak{F} := \{\text{IF \_ THEN \_ ELSE \_} \ll \_; \_ \gg, \emptyset\} \tag{69}$$

$$\mathfrak{H} := \left\{f(\vec{t}) \middle| f \in \mathfrak{F}\right\} \tag{70}$$

Notations 1. *Let S be a set of $\mathcal{E}_0^2$, we write*

$$\boxed{S}_{\mathcal{L}}^{\eta,\rho} := \left\{ [t\{\vec{\imath} \mapsto \sigma(\vec{\imath})\}]^{\mathbb{T}} \middle| \begin{array}{c} (\phi, t) \in S, \\ \vec{\imath} = \text{fv}(\phi) \cup \text{fv}(t), \\ \mathcal{L}^\eta(\phi\{\vec{\imath} \mapsto \sigma(\vec{\imath})\})(\rho) = 1 \end{array} \right\} \tag{71}$$

Where the σ are assignments of the variables.

For a BC term t, let $\mathsf{st}_{\mathscr{BC}}(t)$ be the set of its subterms.

In this section, we will focus on proving our claim that $\sqsubseteq$ really produces an overapproximation of $\mathsf{st}_{\mathscr{BC}}(\_)$. We achieve this by showing Theorem 7.

Theorem 7. *Let $t \in \mathcal{E}_0$, and $\mathcal{L}$ a cryptographic model*

$$\mathsf{st}_{\mathscr{BC}}\left([t]^{\mathbb{T}}\right) \backslash \mathfrak{H} \subseteq \boxed{\mathsf{st}_{\mathcal{P}}(t)}_{\mathcal{L}}^{\eta,\rho} \tag{72}$$

Using monadic notations

$$\mathsf{st}^{i\,c}_{\mathcal{P}}(\mathrm{input}(T)) \coloneqq \{(\mathtt{true}, \mathrm{input}(T))\} \cup \left\{ \begin{array}{l} \mathsf{a}[\vec{\_}] \leftarrow \mathcal{S} \\ \phi \leftarrow \mathtt{true}\ \textbf{if}\ 1 \in \{i, c\}\ \textbf{else}\ \mathsf{a}[\vec{\jmath}] < T \\ m \leftarrow \{\mathrm{msg}(\mathsf{a}[\vec{\jmath}]), \mathrm{cond}(\mathsf{a}[\vec{\jmath}])\} \\ (\_, t) \leftarrow \mathsf{st}^{1\,c}_{\mathcal{P}}(m) \\ \textbf{return}\ \{(\phi, t)\} \end{array} \right\} \tag{65}$$

$$\mathsf{st}^{i\,c}_{\mathcal{P}}(\mathfrak{c}[\vec{\imath}]!(T)) \coloneqq \{(\mathtt{true}, \mathfrak{c}[\vec{\imath}]!(T))\} \cup \left\{ \begin{array}{l} \mathsf{a}[\vec{\_}] \leftarrow \mathcal{S} \\ \phi \leftarrow \mathtt{true}\ \textbf{if}\ 1 \in c, i\ \textbf{else}\ \mathsf{a}[\vec{\jmath}] \leq T \\ (\_, t) \leftarrow \mathsf{st}^{i\,1}_{\mathcal{P}}\left(u_{\mathsf{a}[\vec{\jmath}]}\left(\mathfrak{c}\left[\vec{k}\right]\right)\right) \\ \textbf{return}\ \{(\phi, t)\} \end{array} \right\} \tag{66}$$

$$\mathsf{st}^{i\,c}_{\mathcal{P}}\left(\mathfrak{Q}\vec{\alpha}.\vec{t}\right) \coloneqq \left\{\left(\mathtt{true}, \mathfrak{Q}\vec{\alpha}.\vec{t}\right)\right\} \cup \mathsf{st}^{i\,c}_{\mathcal{P}}\left(\vec{t}\right) \tag{67}$$

$$\mathsf{st}^{i\,c}_{\mathcal{P}}\left(f[\vec{\imath}]\left(\vec{t}\right)\right) \coloneqq \left\{\left(\mathtt{true}, f[\vec{\imath}]\left(\vec{t}\right)\right)\right\} \cup \mathsf{st}^{i\,c}_{\mathcal{P}}\left(\vec{t}\right) \tag{68}$$

Where $\mathfrak{Q}\vec{\alpha}.t_1, t_2, t_3$ stands for FIND $\vec{\alpha}$ SUCH THAT t_1 THEN t_2 ELSE t_3. The $\vec{\imath}$, $\vec{\jmath}$ and $\vec{k}$, when unbound, are fresh variables.
Then $\mathsf{st}_{\mathcal{P}}(u) \coloneqq \mathsf{st}^{0\,0}_{\mathcal{P}}(u)$.

Figure 10: Subterm sets

Using monadic notations

$$\mathsf{st}^{\square}_{\mathcal{P}}(\mathrm{input}(T)) \coloneqq \{(\mathtt{true}, \mathrm{input}(T))\} \cup \left\{ \begin{array}{l} \mathsf{a}[\vec{\_}] \leftarrow \mathcal{S} \\ m \leftarrow \{\mathrm{msg}(\mathsf{a}[\vec{\jmath}]), \mathrm{cond}(\mathsf{a}[\vec{\jmath}])\} \\ (\_, t) \leftarrow \mathsf{st}^{\square}_{\mathcal{P}}(m) \\ \textbf{return}\ \{(\mathsf{a}[\vec{\jmath}] < T, t)\} \end{array} \right\} \tag{73}$$

$$\mathsf{st}^{\square}_{\mathcal{P}}(\mathfrak{c}[\vec{\imath}]!(T)) \coloneqq \{(\mathtt{true}, \mathfrak{c}[\vec{\imath}]!(T))\} \cup \left\{ \begin{array}{l} \mathsf{a}[\vec{\_}] \leftarrow \mathcal{S} \\ (\_, t) \leftarrow \mathsf{st}^{\square}_{\mathcal{P}}\left(u_{\mathsf{a}[\vec{\jmath}]}\left(\mathfrak{c}\left[\vec{k}\right]\right)\right) \\ \textbf{return}\ \{(\mathsf{a}[\vec{\jmath}] \leq T, t)\} \end{array} \right\} \tag{74}$$

$$\mathsf{st}^{\square}_{\mathcal{P}}\left(\mathfrak{Q}\vec{\alpha}.\vec{t}\right) \coloneqq \left\{\left(\mathtt{true}, \mathfrak{Q}\vec{\alpha}.\vec{t}\right)\right\} \cup \mathsf{st}^{\square}_{\mathcal{P}}\left(\vec{t}\right) \tag{75}$$

$$\mathsf{st}^{\square}_{\mathcal{P}}\left(f[\vec{\imath}]\left(\vec{t}\right)\right) \coloneqq \left\{\left(\mathtt{true}, f[\vec{\imath}]\left(\vec{t}\right)\right)\right\} \cup \mathsf{st}^{\square}_{\mathcal{P}}\left(\vec{t}\right) \tag{76}$$

Where $\mathfrak{Q}\vec{\alpha}.t_1, t_2, t_3$ stands for FIND $\vec{\alpha}$ SUCH THAT t_1 THEN t_2 ELSE t_3. The $\vec{\imath}$, $\vec{\jmath}$ and $\vec{k}$, when unbound, are fresh variables.

Figure 11: Temporary Subterm Set

Consider the sets $\mathsf{st}^{\square}_{\mathcal{P}}(\_)$ defined in Fig. 11.

Lemma 3. *Let $t \in \mathcal{E}_0$, and $\mathcal{L}$ a cryptographic model*

$$\mathsf{st}_{\mathscr{BC}}\left([t]^{\mathbb{T}}\right) \backslash \mathfrak{H} \subseteq \boxed{\mathsf{st}^{\square}_{\mathcal{P}}(t)}^{\eta,\rho}_{\mathcal{L}} \tag{77}$$

Proof: By induction over the size of $[t]^{\mathbb{T}}$ and then by case analysis over t. Consider for instance the case $t = \mathrm{input}(T)$. Let $\Delta \coloneqq \mathsf{st}_{\mathscr{BC}}\left([\mathrm{input}(T)]^{\mathbb{T}}\right) \backslash \mathfrak{H}$

$\underline{(T)^{\mathbb{T}} = \mathsf{init}, \mathsf{undef}}$: Then $\Delta = \varnothing$ and we conclude.

$\underline{(T)^{\mathbb{T}} = \mathsf{a}[\vec{\mathbf{1}}]}$: A quick induction shows us that (remember that $\bar{\wedge}$ is simply sugar over an IF):

$$\Delta = \left\{\mathsf{st}_{\mathscr{BC}}\{t'\} \backslash \mathfrak{H} \middle| b <_{\mathbb{S}} \mathsf{a}[\vec{\mathbf{1}}], t' \in \{\mathrm{msg}(b), \mathrm{cond}(b)\}\right\} \tag{78}$$

Then let $u \in \Delta$ with $u \neq \mathrm{input}(T)$. We know that $u \in \mathsf{st}_{\mathscr{BC}}\left([\mathrm{msg}(\mathsf{b}[\vec{\mathbf{j}}])]^{\mathbb{T}}\right)$ (or $\mathrm{cond}(\mathsf{b}[\vec{\mathbf{j}}])$) with $\mathsf{b}[\vec{\mathbf{j}}] \in \mathbb{S}$ such that $\mathsf{b}[\vec{\mathbf{j}}] <_{\mathbb{S}} \mathsf{a}[\vec{\mathbf{1}}]$. That is we can find a assignment σ such that $u \in \mathsf{st}_{\mathscr{BC}}\left([\mathrm{msg}(\mathsf{b}[\sigma(\vec{\jmath})])]^{\mathbb{T}}\right)$ (or $\mathrm{cond}(\mathsf{b}[\sigma(\vec{\jmath})])$). By induction $u \in \boxed{\mathrm{msg}(\mathsf{b}[\vec{\jmath}])}^{\eta,\rho}_{\mathcal{L}}$ and since $\mathcal{L}^{\eta}(\mathsf{b}[\vec{\mathbf{j}}] < \mathsf{a}[\vec{\mathbf{1}}])$, we get $u \in \boxed{\mathsf{st}_{\mathscr{BC}}(\mathrm{input}(T))}^{\eta,\rho}_{\mathcal{L}}$.

Thus we conclude that

$$\Delta \in \boxed{\mathsf{st}_{\mathscr{BC}}(\mathrm{input}(T))}^{\eta,\rho}_{\mathcal{L}} \tag{79}$$

□

Lemma 4.

$$\boxed{\mathsf{st}^{\square}_{\mathcal{P}}(t)}^{\eta,\rho}_{\mathcal{L}} \subseteq \boxed{\mathsf{st}_{\mathcal{P}}(t)}^{\eta,\rho}_{\mathcal{L}} \tag{80}$$

Proof: By induction. The second set is less constraining on the side condition. □

Proof of Theorem 7: By Lemma 3 then 4. □

Finally to make $\mathsf{st}_{\mathcal{P}}(\_)$ fully usable we must ensure it is effectively computable:

Property 14. *For all t, $\mathsf{st}_{\mathcal{P}}(t)$ is finite.*

Proof: This comes from the fact that $\mathsf{st}^{1\,1}_{\mathcal{P}}(\_)$ do not have conditions and are subsets of all the terms appearing in the description of the protocol (which is finite). They are the only recursively defined sets that do not decrease with the structural ordering. □

In practice, we compute $\mathsf{st}_{\mathcal{P}}(\_)$ by memoizing the calls to the inputs and the memory cells. This in turn is equivalent to looking for connected parts of the graph of calls to inputs and memory cells.

Appendix E. Compatibility with squirrel

We remind in Fig. 12 the grammar used in the tool SQUIRREL as defined in [20].

We write $\mathcal{T}^S$ for the set of SQUIRREL terms that can be built out of the non-grayed out grammar from Fig. 12, and $\mathcal{T}^C = \mathbb{S}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ for CRYPTOVAMPIRE's symbolic terms (from Fig. 1). We define $\mathscr{T} : \mathcal{T}^S \mapsto \mathcal{T}^C$ to map between the two. Intuitively, $\mathscr{T}$ commutes with all non-macro operators (including memory cells). Then

$$\mathscr{T}\left(\mathsf{input}@T\right) \coloneqq \mathrm{input}(\mathscr{T}(\underline{T})) \tag{81}$$

Any other term is not supported. This corresponds to at least all non-grayed out terms in Fig. 12. Most of the remaining is recovered using Property 15. Effectively, the only non-supported term is $\mathsf{frame}@T$, which is rarely directly used.

The full description is in Fig. 13.

Lemma 5 (Steps)**.** *Let*

$$\underline{\mathsf{a}[\vec{\imath}]}.\left(\phi_{\mathsf{a}[\vec{\imath}]}, o_{\mathsf{a}[\vec{\imath}]}, \left\{\mathsf{s}[\vec{\jmath}] \leftarrow u_{\mathsf{a}[\vec{\imath}],\mathsf{s}[\vec{\jmath}]} \middle| s \in \mathcal{C}\right\}\right)$$

$$
\begin{aligned}
T &:= \tau \mid \mathsf{a}[\vec{\imath}] \mid \mathsf{pred}(T) \\
t &:= x \mid \mathsf{c}[\vec{\imath}]@T \mid \mathsf{n}[\vec{\imath}] \mid f[\vec{\imath}](\vec{t}) \\
&\quad \mid \mathsf{input}@T \mid \mathsf{output}@T \mid \mathsf{frame}@T \\
&\quad \mid \mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t' \\
&\quad \mid \mathsf{find}\ \vec{\imath}\ \mathsf{such\ that}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t' \\
A &:= t = t' \mid i = i' \mid T = T' \mid T < T' \mid T \leq T' \\
&\quad \mid \mathsf{happens}(T) \mid \mathsf{cond}@T \mid \mathsf{exec}@T \\
\phi &:= A \mid \top \mid \bot \mid \phi \land \phi' \mid \phi \lor \phi' \mid \phi \Rightarrow \phi \mid \neg\phi \\
&\quad \mid \forall i.\phi \mid \exists i.\phi \mid \exists \tau.\phi \mid \forall \tau.\phi
\end{aligned}
$$

Figure 12: SQUIRREL [20]'s syntax

be a SQUIRREL *action, then, when applicable,*

$$\mathsf{a}[\vec{\imath}] := \left(\mathscr{T}(\phi_{\mathsf{a}[\vec{\imath}]}), \mathscr{T}(o_{\mathsf{a}[\vec{\imath}]}), \lambda \mathsf{c}\left[\vec{j}\right].\mathscr{T}(u_{\mathsf{a}[\vec{\imath}],\mathsf{c}[\vec{j}]})\right)$$

is a CRYPTOVAMPIRE *step.*

Using the λ notation to define functions.

We will then assume CRYPTOVAMPIRE and SQUIRREL share their set of step/action names $\mathcal{S}$.

Lemma 6 (Protocol)**.** *Let $\underline{P} = (P_{\text{act}}, \mathcal{U}_0, <)$ be a* SQUIRREL *protocol, then $\mathcal{P} = (\{\mathsf{a}[\_] | \mathsf{a}[\vec{\imath}] \in P_{\text{act}}\}, <)$ is a* CRYPTOVAMPIRE *abstract protocol.*

Proof: $<$ is a partial order over $\{\mathsf{a}[\vec{\imath}] | \mathsf{a}[\_] \in S\}$, thus it is a preorder. Moreover, $<$ is insensitive to the indices. Finally, all steps may only refer to previous steps, and memory cells do not have cyclic calls (as they only refer to previous steps). □

Lemma 7 (Trace)**.** *Let $\mathbb{T}_{\text{sq}} = (\mathbb{I}, \mathcal{D}_{\mathcal{T}}^{\text{sq}}, <_T, \sigma_{\mathbb{I}}, \sigma_T)$ be a* SQUIRREL *trace. Then $\mathbb{T} = (\mathbb{I}, \mathcal{D}_{\mathcal{T}}^{\text{sq}}, \sigma_{\mathbb{I}}, <_T)$ is a* CRYPTOVAMPIRE *trace.*

We write $[\_]_{\text{sq}}^{\mathbb{T}}$ for the SQUIRREL expansion.

Theorem (Interoperability (1))**.** *For any computational model*[4]*,* SQUIRREL *protocol (and its* CRYPTOVAMPIRE *variant), trace $\mathbb{T}$ over it, security parameter η and random tapes ρ, we have*

$$\mathcal{L}^\eta(|\mathscr{T}(\underline{t})|)(\rho) = \left[\!\left[[t]_{\text{sq}}^{\mathbb{T}} \right]\!\right]_{bc}^{\mathbb{M}}(1^\eta, \rho) \tag{22}$$

Proof: By induction on the size of $[t]_{\text{sq}}^{\mathbb{T}}$ where Property 9 along with Lemma 2 and Property 4 tells us that the unfolding of the quantifiers in CRYPTOVAMPIRE and SQUIRREL can be assumed to be the same. □

Property 15 (Translation of marcos)**.** *In* SQUIRREL

$$\vdash \mathsf{cond}@\tau \Leftrightarrow \bigvee_{\mathsf{a}[\vec{\imath}] \in P_{\text{act}}} \exists \vec{\imath}.\tau = \mathsf{a}[\vec{\imath}] \land \mathsf{cond}@\mathsf{a}[\vec{\imath}] \tag{82}$$

$$\vdash \mathsf{exec}@T \Leftrightarrow (\forall \tau.\tau \leq T \Rightarrow \mathsf{cond}@T) \tag{83}$$

[4]The notion is the same for CRYPTOVAMPIRE and SQUIRREL as it comes from the BC Logic and they share their function symbols

This lets us effectively use $\mathsf{exec}@T$ *in* CRYPTOVAMPIRE.

Proof: (82) by exhaustiveness of the steps and (83) by induction on T. □

Appendix F. Cryptographic Axioms

In this section, we present the other cryptographic notions we support, along with a proof sketch of their axiom.

F.1. No Guessing Theorem

In this section, we produce a proof for Theorem 2.

Theorem (2 No Guessing)**.** *It is not possible to guess honest nonces. Formally, for all nonce $\mathsf{n}[\_] \in \mathcal{N}$, ground indices $\vec{\mathtt{1}} \in \mathcal{I}$, and message m, we have*

$$|\mathsf{n}[\vec{\mathtt{1}}]| = |m| \Rightarrow \mathsf{n}[\vec{\mathtt{1}}] \sqsubseteq m \tag{23}$$

Proof: Let $\mathsf{n} \in \mathcal{N}$ and nonce name $\vec{\mathtt{1}}$ some indices and m a message.

Suppose $\mathcal{L}^\eta(\mathsf{n}[\vec{\mathtt{1}}] \sqsubseteq m)(\rho) = 0$ (notice that this then holds for all ρ). By Theorem 7 we get that the random variables $\mathcal{L}^\eta(\mathsf{n}[\vec{\mathtt{1}}])$ and $\mathcal{L}^\eta(m)$ are independants as m never gets access to the relevant part of ρ_h.

Thus

$$\mathbb{P}\text{rob}_\rho(\mathcal{L}^\eta(\mathsf{n}[\vec{\mathtt{1}}] = m)) = \frac{1}{2^\eta} = \text{negl}(\eta) \tag{84}$$

□

We continue with a proof of Section 5.3.2's version of Theorem 2.

Theorem (4)**.** *For all nonces $\mathsf{n} \in \mathcal{N}$,* (37) *can be added to the set of axioms*

$$\forall \vec{\imath}, m.\ |\mathsf{n}[\vec{\imath}]| = |m| \Rightarrow \mathsf{n}[\vec{\imath}] \sqsubseteq^\circ m \tag{37}$$

where $\sqsubseteq^\circ$ is defined by (38) *and is compatible with $\doteq$.*

$$\mathsf{m} \sqsubseteq^\circ |t| := \forall t'.(|t| = |t'| \Rightarrow \mathsf{m} \sqsubseteq t'). \tag{38}$$

Proof: Let $\forall \vec{u}.\Gamma$ be an axiom containing Theorem 2, that is such that

$$\vDash (\forall \vec{u}.\Gamma) \Rightarrow \forall \vec{\imath}, m.\ |\mathsf{n}[\vec{\imath}]| = |m| \Rightarrow \mathsf{n}[\vec{\imath}] \sqsubseteq m \tag{85}$$

Then by transitivity of $=$, we have

$$\vDash (\forall \vec{u}.\Gamma) \Rightarrow (37) \tag{86}$$

Thus, for all t, a proof of $\vDash (\forall \vec{u}.\Gamma) \Rightarrow |t|$ implies a proof of $\vDash (\forall \vec{u}.\Gamma) \land (37) \Rightarrow |t|$. Therefore, Theorem 3 lets us add (37) to the axiom pool while retaining the cryptographic semantics of the proof (as long as Γ is a described in Theorem 3).

Then the claimed compatibility with $\doteq$ comes from (87).

$$\vDash \forall t, t'.\ \mathsf{m} \sqsubseteq^\circ t \land |t| = |t'| \Rightarrow \mathsf{m} \sqsubseteq^\circ t' \tag{87}$$

The nonce itself can be extracted from the reasoning modulo $\doteq$ using a type system as described in Appendix G.1. □

$$\mathscr{T}(\underline{\tau}) := \tau \qquad \mathscr{T}(\underline{i}) := i \qquad \mathscr{T}\left(\underline{\mathsf{a}[\vec{\imath}]}\right) := \mathsf{a}[\vec{\imath}] \qquad \mathscr{T}\left(\underline{\mathrm{pred}(T)}\right) := \mathrm{pred}(\mathscr{T}(\underline{T}))$$

$$\mathscr{T}\left(\underline{\mathsf{happens}(T)}\right) := \mathrm{happens}(\mathscr{T}(\underline{T})) \qquad \mathscr{T}\left(\underline{\mathsf{c}[\vec{\imath}]@T}\right) := \mathsf{c}[\vec{\imath}](\mathscr{T}(\underline{T})) \qquad \mathscr{T}\left(\underline{\mathsf{n}[\vec{\imath}]}\right) := \mathsf{n}[\vec{\imath}]$$

$$\mathscr{T}\left(\underline{f[\vec{\imath}](t_1,\ldots,t_n)}\right) := f[\vec{\imath}](\mathscr{T}(\underline{t_1}),\ldots,\mathscr{T}(\underline{t_n})) \qquad \mathscr{T}\left(\underline{\mathsf{input}@T}\right) := \mathrm{input}(\mathscr{T}(\underline{T})) \qquad \mathscr{T}\left(\underline{\mathsf{output}@\mathsf{a}[\vec{\imath}]}\right) := \mathrm{msg}(\mathsf{a}[\vec{\imath}])$$

$$\mathscr{T}\left(\underline{\mathsf{cond}@\mathsf{a}[\vec{\imath}]}\right) := \mathrm{cond}(\mathsf{a}[\vec{\imath}]) \qquad \mathscr{T}(\underline{t = t'}) := \mathscr{T}(\underline{t}) \equiv \mathscr{T}(\underline{t'}) \qquad \mathscr{T}(\underline{i = i'}) := \mathscr{T}(\underline{i}) = \mathscr{T}(\underline{i'})$$

$$\mathscr{T}(\underline{T = T'}) := \mathscr{T}(\underline{T}) = \mathscr{T}(\underline{T'}) \qquad \mathscr{T}(\underline{T < T'}) := \mathscr{T}(\underline{T}) < \mathscr{T}(\underline{T'}) \qquad \mathscr{T}(\underline{\top}) := \mathtt{true} \qquad \mathscr{T}(\underline{\bot}) := \mathtt{false}$$

$$\mathscr{T}(\underline{\phi \Box \phi}) := \mathscr{T}(\underline{\phi}) \,\bar{\Box}\, \mathscr{T}(\underline{\phi}) \qquad \mathscr{T}(\underline{\neg\phi}) := \bar{\neg}\mathscr{T}(\underline{\phi}) \qquad \mathscr{T}(\underline{\mathfrak{Q}\vec{\imath},\vec{\tau}.\phi}) := \bar{\mathfrak{Q}}\vec{\imath},\vec{\tau}.\mathscr{T}(\underline{\phi})$$

$$\mathscr{T}(\underline{\mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t'}) := \textsc{If}\ \mathscr{T}(\underline{\phi})\ \textsc{Then}\ \mathscr{T}(\underline{t})\ \textsc{Else}\ \mathscr{T}(\underline{t'})$$

$$\mathscr{T}(\underline{\mathsf{find}\ \vec{\imath}\ \mathsf{such\ that}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t'}) := \textsc{Find}\ \vec{\imath}\ \textsc{Such that}\ \mathscr{T}(\underline{\phi})\ \textsc{Then}\ \mathscr{T}(\underline{t})\ \textsc{Else}\ \mathscr{T}(\underline{t'})$$

where $\Box \in \{\wedge, \vee, \Rightarrow\}$ and $\mathfrak{Q} \in \{\forall, \exists\}$.

Figure 13: Mapping between SQUIRREL's terms and CRYPTOVAMPIRE's

F.2. Euf-Cma

We already presented the axiom of MACs in Property 1. We start from the BC rule from Property 3.

Proof of Property 3: We refer to previous work [19] for a more in-depth proof.

The intuition is that assuming (14) does not hold, then (m, σ) is a winning attacker to the EUF-CMA security game, which contradicts the assumptions. □

We can then lift to Property 1.

Proof of Property 1: We start from the BC formula:

$$\mathcal{C}_{\text{EUF-CMA}} \vDash_{\text{bc}} \mathsf{verify}(\sigma, m, \mathsf{k}) \Rightarrow \bar{\bigvee}_{\mathcal{H}(u,\mathsf{k}) \in \mathsf{st}_{\mathscr{BC}}(m,\sigma)} u \equiv m \tag{14}$$

By Theorem 7 and Property 5 we get that

$$\mathcal{C}_{\text{EUF-CMA}} \vDash_{\text{bc}} \mathsf{verify}(\sigma, m, \mathsf{k}) \Rightarrow \bar{\bigvee}_{\mathcal{H}(u,\mathsf{k}) \in \mathsf{st}_{\mathcal{P}}(m,\sigma)} u \equiv m \tag{88}$$

Turing (88) into a formula of $\mathscr{L}(\mathcal{N}, \mathcal{F}, \mathcal{I}, \mathcal{C}, \mathcal{S})$ and using Property 4, we get

$$\mathcal{C}_{\text{EUF-CMA}} \vDash_{\mathbb{P}} |\mathsf{verify}(\sigma, m, \mathsf{k})| \Rightarrow \bigvee_{\mathcal{H}(u,\mathsf{k}) \in \mathsf{st}_{\mathcal{P}}(m,\sigma)} |u| = |m| \tag{89}$$

Which in turn (including the side condition in the formula) gives us (2) for Property 1. □

Very much the same way we show the public key version of EUF-CMA:

Property 16 (Euf-Cma)**.** *When* sign*,* verify *and* vk *form a signature scheme that is existentially unforgeable under chosen message attacks (*EUF-CMA*), then the protocol $\mathcal{P}$ satisfies the following:*

$$|\mathsf{verify}(\sigma, m, \mathsf{vk}(\bar{\mathsf{k}}))| \Rightarrow \begin{pmatrix} \mathsf{k} \sqsubseteq_{\mathsf{sign}(\_,\bullet),\mathsf{vk}(\bullet)} m, \sigma, \mathcal{P} \\ \vee\, \exists u. (\mathsf{sign}(u, \bar{\mathsf{k}}) \sqsubseteq m, \sigma \wedge |u| = |m|) \end{pmatrix} \tag{90}$$

where k *is a* Nonce*.*

Since the subterm relation is a bounding formula (Definition 8), Properties 6 and 7 tell us that we can expand Property 1 and Property 16 from axioms schemas into formula with bounded Skolem normal form.

F.3. Int-Ctxt

Property 17 (Int-Ctxt)**.** *When* senc *and* dec *form a symmetric encryption scheme that conserves the integrity of the ciphertext (*INTCTXT*), any protocol $\mathcal{P}$ verifies:*

$$\begin{aligned} &|\mathsf{verify}(c, \bar{\mathsf{k}})| \Rightarrow \\ &\quad \exists m, r. (|\mathsf{senc}(m, r, \bar{\mathsf{k}})| = |c| \wedge \mathsf{senc}(m, r, \bar{\mathsf{k}}) \sqsubseteq c) \\ &\qquad \vee\, \mathsf{k} \sqsubseteq'_{\mathsf{senc}(\_,\_,\bullet),\mathsf{sdec}(\_,\bullet),\mathsf{verify}(\_,\bullet)} c, \mathcal{P} \\ &\qquad\qquad \vee\, \neg\mathsf{senc\text{-}rand}^{\mathcal{P}}_{\mathsf{k}}(u) \end{aligned} \tag{91}$$

where k *is a* Nonce *and* $\mathsf{verify}(c, k) := \mathsf{dec} \not\equiv \mathsf{fail}$.

$\mathsf{senc\text{-}rand}^{\mathcal{P}}_{\mathsf{k}}(u)$ *holds when in all instances of* $c_r = \mathsf{senc}(m, r, \bar{\mathsf{k}})$ *appearing in the extension of* u *in any trace,* r *is of the form* $\bar{\mathsf{r}}$ *and only appear in* u *in* c_r*.*

Proof: This is a similar lifting of a rule from [19]. □

Appendix G.
Engineering Details

G.1. Effective First-Order Representation

As presented in Section 4.1, the main first-order formalization challenges revolve around the interactions (or lack thereof) between $\doteq$ and $\sqsubseteq$.

Therefore, efficient first-order reasoning within CRYPTOVAMPIRE crucially depends on built-in, native support for **(i)** efficiently distinguishing when to use $=$ and $\doteq$ in order to quickly evaluate the subterm relation (e.g., inferring that x is a subterm of $x \,\bar{\wedge}\, y$). Additionally, **(ii)** base formulas should also be handled efficiently (e.g., $\bar{\wedge}$ should be treated as a logical conjunction).

We resolve these issues in CRYPTOVAMPIRE, by (i) considering multi-sorted first-order representations of Section 4, for which we propose extensions to the SMT-LIB type system [40]. To this end, we declare symbolic terms (t in Fig. 1) as *datatypes*, allowing us to distinguish between three main types: Message (symbolic bitstring computations that can be sent over the network), Condition (symbolic boolean computations), and Nonce. We introduce the function $\bar{\mathsf{n}}$ to map Nonce to Message, enabling us to "distinguish" honest nonces in cryptographic axioms and speed up reasoning with $=$ using types. Similarly, we enumerate possible CRYPTOVAMPIRE steps via datatypes, introducing the type Time.

Further, symbolic quantifiers and lookups (e.g., $\bar{\exists} i.\phi$) are named in the following sense: for each term t denoting a symbolic quantifier $\mathfrak{Q}\vec{x}.\vec{y}$, where $\mathfrak{Q}$ is $\bar{\forall}$, $\bar{\exists}$, or FIND _ SUCH THAT _ THEN _ ELSE _, we introduce a new honest function $f_{\mathfrak{Q}}$. Here, the function $f_{\mathfrak{Q}}$ takes the free variables of t as its arguments. We additionally consider the respective axioms for $|f_{\mathfrak{Q}}(\dots)|$ and subterm relations, such as instances of axiom (34).

For efficient reasoning modulo $\doteq$, (ii) we introduce the new sort Bitstring and, for each $f[\vec{\_}] \in \mathcal{F}$, we consider a free function $|f|$ such that

$$|f[\vec{\imath}](t_1, \dots, t_n)| = |f|(\vec{\imath}, |t_1|, \dots, |t_n|) \qquad (92)$$

As such, Bitstring is used to denote the sort of evaluated Message, whereas the built-in Bool represents the sort of evaluated Condition. CRYPTOVAMPIRE's Evaluated Logic (Section 4.2) is represented as is using standard Boolean connectives.

The above considerations (i)–(ii) provide first-order encodings for CRYPTOVAMPIRE formalization. To turn reasoning over such encodings efficient, we introduce various modifications to the saturation-based theorem proving over CRYPTOVAMPIRE encodings, complemented with preprocessing heuristics.

G.2. Customized Subterm Relations

Unfortunately, this general subterm relation $\sqsubseteq$ is often not expressive enough for our class of problems. Already in the EUF-CMA axiom (Property 1 of Section 2), we use $\sqsubseteq_{\mathsf{verify}(\_,\_,\bullet),\mathcal{H}(\_,\bullet)}$. This means we ignore the positions of the key when looking for subterms through $\mathcal{H}(\_,\_)$ and $\mathsf{verify}(\_,\_,\_)$.

In the general case, we write $\sqsubseteq_{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)}$ for some $f[\vec{\_}] \in \mathcal{F}$ to express that one should ignore the j^{th} argument when looking through f. Such a relation cannot be expressed solely through the base $\sqsubseteq$. Therefore, we introduce it as a brand-new binary relation that closely resembles $\sqsubseteq$. CRYPTOVAMPIRE computes new sets $\mathsf{st}_{\mathcal{P}}^{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)}(\_)$ and we describe the relation to the theorem prover by adapting the axioms of Fig. 4 to $\mathsf{st}_{\mathcal{P}}^{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)}(\_)$. In particular, the instance of axiom (33) for f is replaced by:

$$\begin{aligned}&\forall t, t_1, \dots, t_n, \vec{\imath}.\\ &t \sqsubset_{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)} f[\vec{\imath}](t_1, \dots, t_n) \wedge t = t_j \Rightarrow\\ &\qquad \bigvee_{\substack{k=1\\k\neq j}}^{n} t \sqsubseteq_{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)} t_k\end{aligned} \qquad (93)$$

$$\begin{aligned}&\forall t, t_1, \dots, t_n, \vec{\imath}.\\ &t \sqsubset_{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)} f[\vec{\imath}](t_1, \dots, t_n) \wedge t \neq t_j \Rightarrow\\ &\qquad \bigvee_{k=1}^{n} t \sqsubseteq_{f[\vec{\_}](\_1,\dots,\bullet_j,\dots,\_n)} t_k\end{aligned} \qquad (94)$$

Example 19 gives an example of such a customized subterm relation.

Example 19. *With the simpler case of* $\sqsubseteq_{\mathcal{H}(\_,\bullet)}$*, we replace the instance of Equation 33 where* f *is* $\mathcal{H}$ *with*

$$\forall t, m.\ t \sqsubset_{\mathcal{H}(\_,\bullet)} \mathcal{H}(m, t) \Rightarrow t \sqsubseteq_{\mathcal{H}(\_,\bullet)} m \qquad (95)$$

$$\begin{aligned}&\forall t, m_1, m_2.\\ &\big(t \sqsubset_{\mathcal{H}(\_,\bullet)} \mathcal{H}(m_1, m_2) \wedge t \neq m_2\big) \Rightarrow \bigvee_{k=1}^{2} t \sqsubseteq_{\mathcal{H}(\_,\bullet)} m_k\end{aligned} \qquad (96)$$

Such customization naturally extends to more than one function and more than one argument position.

G.3. Forcing Rewritings

Saturation-based provers rely on term orderings to keep their proof search small [41], by ensuring that smaller terms (w.r.t. the ordering) are not rewritten by equal larger terms. To further reduce the application of equality (and thus rewritings of equal terms), we use equality reasoning based on $\doteq$ by using the Bitstring sort from Section G.1. Doing so, we orient the axiom (92) as

$$\left|f\left[\vec{I}\right](t_1, \dots, t_n)\right| \overset{\rightarrow}{\doteq} |f|\big(\vec{I}, |t_1|, \dots, |t_n|\big) \qquad (92)$$

where $\overset{\rightarrow}{\doteq}$ denotes that $\left|f\left[\vec{I}\right](t_1, \dots, t_n)\right|$ is bigger than (and thus should be rewritten by) $|f|\big(\vec{I}, |t_1|, \dots, |t_n|\big)$. Such an orientation goes, however, against standard orderings, as, for example, constants/unary functions are usually smaller than function symbols of those of higher arity.

To enforce (92), we introduce additional orderings over CRYPTOVAMPIRE terms; while these extensions may not preserve refutational completeness, our experiments show overall good performance (see Section 6).