Automated Reasoning for Indistinguishability in the CCSA

Simon Jeanteur
TU Wien
Vienna, Austria
simon.jeanteur@tuwien.ac.at

Laura Kovács
TU Wien
Vienna, Austria
laura.kovacs@tuwien.ac.at

Matteo Maffei
TU Wien
Vienna, Austria
matteo.maffei@tuwien.ac.at

Michael Rawson
University of Southampton
Southampton, United Kingdom
michael@rawsons.uk

Abstract

Cryptographic protocols are the foundation of secure digital communication, yet their design remains error-prone, as evidenced by the vulnerabilities that have plagued even the most widely adopted protocols throughout history. Security properties are typically formalized using either trace properties or indistinguishability, each addressing distinct security guarantees, such as agreement and authenticity for the former and anonymity and strong secrecy for the latter. Formal verification of cryptographic protocols spans both symbolic and computational models. While symbolic techniques enable automation and scalability, they do not provide computational security guarantees. Computational models, though robust, are harder to formalize and automate. Recent advances, such as the *Computationally Complete Symbolic Attacker* (CCSA) model and its logic, the *Bana-Comon Logic* (BC Logic), bridge this gap by supporting both trace properties and indistinguishability. However, despite significant progress in proof assistants, automating indistinguishability remains a challenge due to its combination of unstructured equality theories, complex non-classical calculus, and partially inductive reasoning—all requiring expert knowledge in both cryptography and logic.

This paper introduces a novel approach to automate indistinguishability proofs in the CCSA model, implemented in the automated prover CryptoVampire2. We extend CryptoVampire to support indistinguishability by designing golgge, a Prolog-inspired backtracking engine over equality graphs (e-graphs), which provides strong, rewrite-driven equational reasoning capabilities. We adapt the BC Logic rules to this new framework, yielding semantically compatible statements. The effectiveness of our approach is demonstrated by automating all indistinguishability goals in the Squirrel repository.

CCS Concepts

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

CCS '26, The Hague, Netherlands

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846552

Keywords

Security Protocols, Indistinguishability, Formal Methods, Computational Security, Automated Theorem Proving

ACM Reference Format:
Simon Jeanteur, Laura Kovács, Matteo Maffei, and Michael Rawson. 2026. Automated Reasoning for Indistinguishability in the CCSA. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846552

1 Introduction

Cryptographic protocols serve as the foundation enabling secure communication among the components of our digital infrastructure. However, their design remains a challenging and error-prone endeavor. Historically, vulnerabilities have been uncovered in widely adopted protocols, ranging from the classic attack on the Needham-Schroeder protocol [54] to repeated breaches in the TLS protocol [3], and even in emerging blockchain protocols [56]. These incidents underscore the need for rigorous verification methods.

Formal methods have emerged as a robust approach to ensure the correctness of cryptographic protocols. Notably, they have played a pivotal role in the development of modern standards such as TLS 1.3 [15], WireGuard [52], and 5G-AKA [50]. Security properties for these protocols are typically formalized using either *trace properties* [55] or *indistinguishability relations* [25]. Trace properties focus on guarantees for individual execution traces, encompassing secrecy, authentication, and constraints on the order of security-relevant events. Indistinguishability, on the other hand, ensures that an attacker cannot distinguish between two protocol configurations, such as identifying which of two secrets is in use.

1.1 Related Work

Formal verification of cryptographic protocols is broadly categorized into *symbolic* and *computational* models. Symbolic techniques abstract away from the algebraic properties of cryptography, modeling attackers as entities with infinite computational resources but limited to constructing symbolic terms from their existing knowledge (the "Dolev-Yao" attacker [35]). This abstraction simplifies proof techniques and has led to the development of automated verifiers like ProVerif [17] and Tamarin [58]. However, symbolic security does not guarantee computational security; protocols proven secure in the symbolic model may still be vulnerable to attacks exploiting weaknesses in their cryptographic implementations [16].

Conversely, the computational model uses probabilistic and computational notions to mirror real-world implementations. Here, attackers are probabilistic polynomial-time Turing machines, and proofs reduce protocol security to mathematical assumptions. While providing strong guarantees, this model is harder to formalize and automate. Nonetheless, several tools have emerged, attempting automation. For instance, EasyCrypt [14] is a cryptographic proof checker simplifying reasoning about probabilistic computations, typically used for cryptographic schemes. Additionally, CryptoVerif [18] automates game-based proofs for protocols but lacks support for states. Finally, Owl [38] uses types for integrity and confidentiality properties.

Recent efforts have focused on bridging the gap between symbolic and computational models, seeking to combine the ease of symbolic proofs with the strong guarantees of the computational model. A pivotal development in this direction is the *Computationally Complete Symbolic Attacker* (CCSA) model and its associated logic, the Bana-Comon (BC) Logic [9–11] , which has since been formalized as a fragment of higher-order logic [29]. The BC Logic supports both trace properties and indistinguishability (Def. 1), enabling computationally sound formal verification. While early proofs in the BC Logic were conducted manually, recent advancements have led to the development of the interactive proof assistant Squirrel [5] and the automated prover Cryptovampire [45], the latter supporting only trace properties. The growing interest in CCSA has further extended its capabilities to include stateful protocols [6], post-quantum security [28], and proof composition [27]. Squirrel, as an interactive theorem prover, has made significant progress in usability through custom security games [8], automation of non-cryptographic goals [7], and strong typing [34] . However, it still demands expertise in both cryptography and logic, and full automation, in particular of indistinguishability proofs, remained until now an open problem.

Automating indistinguishability goals poses a number of challenges. Proofs in this setting typically require complex equality reasoning in a non-classical logic, necessitating non-trivial choices from cryptographers when applying cryptography-related rules. These challenges can in principle be addressed individually: congruence closure algorithms [67] handle equational reasoning; some theorem provers can reason in non-classical logics [62]; undirected choice can be modeled as SAT and related approaches [30, 57]; and Prolog [26] and Datalog [22] execute rule systems. However, integrating all these requirements into one system is an open challenge. *Satisfiability Modulo Theory* (SMT) solvers [12, 33, 51] are natural candidates, given their ability in principle to handle all these requirements (including non-classical logic [36]), but they already proved to require external assistance for trace properties [45] and remain incapable of handling indistinguishability goals independently, especially in quantified theories. A fundamentally novel approach is thus needed for machine-checked, automated reasoning about indistinguishability properties.

1.2 Our Contributions

This paper introduces the first methodology for automating indistinguishability proofs in the BC Logic. Our approach is implemented in new version of CryptoVampire[1], called CryptoVampire2, and delivers the following key contributions:

(A) Leveraging automated equality reasoning in the form of egg [67] and inspired by egglog [68], we introduce golgge, a Prolog-like [26] backtracking engine reasoning modulo equality.

(B) We adapt Squirrel's rules to the golgge engine, rethinking the subterm relations ubiquitous in the CCSA to align with equality reasoning. We also extend the concept of *bi-deducibility* [6] to avoid expensive rules. We ensure compatibility with Squirrel's first-order semantics and prove the soundness of our approach.

(C) We conduct an *experimental evaluation* of CryptoVampire2 on challenging, state-of-the-art protocols from [5, 6], in particular, we consider all indistinguishability queries in the Squirrel protocol library that are expressible as bi-processes (i.e., pairs of protocols sharing a structure, but differing on terms), which is the format that our tool supports at present.[2] CryptoVampire2 automates all such queries, taking as input the protocol specification and the security goals, expressed as trace properties, in less than 2 min 15 s each, on consumer hardware, showcasing its efficiency as a verifier.

The technical report [46] provides further details and proofs.

2 Overview

CryptoVampire2 attempts to prove the indistinguishability of user-specified protocols. To that end, CryptoVampire2 expects the following inputs from its users: **(i)** function declarations, **(ii)** steps representing atomic inputs/outputs, **(iii)** their instantiations for each protocol, and **(iv)** constraints on all the above. These constraints could be cryptographic assumptions, simplification rules, structural dependencies across steps, or supplementary lemmas. CryptoVampire2 then attempts to prove the indistinguishability of the specified protocols by building a proof tree using the BC Logic and Squirrel-compatible rules.

Example 1 (Basic Hash). *To illustrate, we introduce the classic CCSA example in the form of the following RFID-like protocol [20]:*

$$\begin{aligned} T_j \to R: &\quad \langle \mathsf{n}, \mathcal{H}(\mathsf{n}, \mathsf{k}_j) \rangle \\ \leftarrow R: &\quad \mathsf{ok} \ \textit{if} \ \mathsf{k}_j \ \textit{is in } R\textit{'s database} \end{aligned} \tag{1}$$

Here, a tag T_j outputs a fresh *nonce* n *both in plaintext 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). This protocol features* unlinkability*, meaning an attacker cannot trace a tag across multiple executions. This is expressed by observing that* (1) *is indistinguishable from:*

$$\begin{aligned} T_j \to R: &\quad \langle \mathsf{n}, \mathcal{H}(\mathsf{n}, \mathsf{k}'_{ij}) \rangle \\ \leftarrow R: &\quad \mathsf{ok} \ \textit{if} \ \mathsf{k}'_{ij} \ \textit{is in } R\textit{'s database} \end{aligned} \tag{2}$$

where i identifies multiple runs of the protocol. In (1)*, k_j is reused across sessions, while* (2) *uses a fresh key each time.*

We now illustrate the main steps of analyzing Ex. 1 in CryptoVampire2.

[1] The old version of the tool will be referred to by its paper, i.e., [45].
[2] This notably includes *all* examples from the original [5].

2.1 Formalization of the Protocol

Following the CCSA model, cryptography is represented *symbolically* using a term algebra 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 $\mathcal{F}$ become *uninterpreted functions* in CryptoVampire2, representing deterministic polynomial-time Turing machines.

Example 2 (Functions). *To encode Ex. 1, we use the following functions in* $\mathcal{F}$*:* $\langle \_, \_ \rangle, \pi_1(\_), \pi_2(\_)$ *for pair construction and projection;* ok, ko *for success and failure of tag authentication;* $\mathcal{H}(\_, \_)$ *for hash computation; and* **If** _ **then** _ **else** _ *for conditionals.*

2.1.2 Honest Randomness. The only source of honest randomness in CryptoVampire2 is *nonces*, representing pairwise independent uniformly random bitstrings of length η, the security parameter. In the logic, nonces are constants used to express cryptographic properties (e.g., collision-resistance). Nonces are indexed to support freshness across an unbounded number of sessions and participants.

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

2.1.3 Protocol Steps. Within CryptoVampire2, a protocol is a set of atomic *steps*, and each execution is a valid sequence of those steps. A step takes an *input* and computes an *output* (or *message*). It may be guarded by a *condition*, specifying when its execution succeeds. To consider indistinguishability, protocols must share some common structure to enable lock-step execution similar to ProVerif's Bi-Processes [19]. In the context of the CCSA we say such protocols are *compatible*, meaning they share their step identifiers.

Example 4 (Steps). *We express Ex. 1's* (1) *and* (2) *using the following steps.* p *is the protocol (*(1) *or* (2)*), and* mk *is an alias for the keys:* $\mathrm{mk}(i,j,(1)) := \mathsf{k}[j]$ *and* $\mathrm{mk}(i,j,(2)) := \mathsf{k}'[i,j]$.

$\underline{\mathsf{T}[i,j]}$*: (*$i^{th}$ *execution of the* j^{th} *tag)*

***Condition**:* true

***Message**:* $\langle \mathsf{n}[i,j], \mathcal{H}(\mathsf{n}[i,j], \mathrm{mk}(i,j,p)) \rangle$

$\underline{\mathsf{Rs}[i,j]}$*: (Authentication of the* j^{th} *tag on the* i^{th} *execution)*

***Condition**:* $\pi_2(\mathsf{in}) \doteq \mathcal{H}(\pi_1(\mathsf{in}), \mathrm{mk}(i,j,p))$

***Message**:* ok

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

$\underline{\mathsf{Rf}[l]}$*: (No authentication on the* l^{th} *execution)*

***Condition**:* $\neg\bar{\exists} i, j.\pi_2(\mathsf{in}) \doteq \mathcal{H}(\pi_1(\mathsf{in}), \mathrm{mk}(i,j,p))$

***Message**:* ko

where in *stands for* $\mathsf{input}@_p\mathsf{Rf}[l]$.

Protocol steps are executed following an order captured by the relation $\mathsf{a} < \mathsf{b}$. We also introduce the mutually exclusive relation $\diamond$ to ensure at most one can be executed in the same protocol run.

Example 5 (Mutual Exclusion). *In Ex. 1, the reader can only execute one branch per session. Thus, for each session* i*, the authentication either succeeds or fails, and at most one tag can be authenticated:*

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

No other ordering constraints apply to this protocol.

2.1.4 Semantic and Cryptographic Properties. The semantics of constants and functions are expressed through rewriting rules, inference rules, and first-order formulas. Following the CCSA model, these are proven against the computational attacker and relevant cryptographic assumptions, capturing what the attacker *cannot* do.

The simplest properties are *directed equality rules*, used by Golgge to manipulate proof goals before applying an inference rule.

Example 6 (Projection). π_1 *is a destructor for* $\langle \_, \_ \rangle$*, yielding the following rewrite rule:*

$$\pi_1(\langle x, y \rangle) \rightsquigarrow x \tag{4}$$

These rules also encode cryptographic and protocol-specific properties. For instance, Lem. 1 describes Ex. 1's authentication.

Lemma 1 (Authentication). *Both protocols of Ex. 1 manifest* authentication *between tags and the reader: when* happens $(\tau) \doteq \mathsf{true}$*,*

$$\pi_2(\mathsf{input}@_p\tau) \doteq \mathcal{H}(\pi_1(\mathsf{input}@_p\tau), \mathrm{mk}(i,j,p)) \rightsquigarrow \begin{pmatrix} \bar{\exists} i'.\mathsf{T}[i',j] < \tau \\ \wedge\ \pi_1(\mathsf{input}@_p\tau) \doteq \pi_1(\mathsf{msg}@_p\mathsf{T}[i',j]) \\ \wedge\ \pi_2(\mathsf{input}@_p\tau) \doteq \pi_2(\mathsf{msg}@_p\mathsf{T}[i',j]) \end{pmatrix} \tag{5}$$

for any τ*. This means that if the condition of a step* Rs *succeeds (i.e., a reader believes authentication succeeded) then a corresponding step* T *occurred earlier, and the inputs and outputs of both steps matched (i.e., the corresponding tag indeed tried to authenticate).*

Properties that cannot be expressed as equality rules, typically related to cryptographic properties, are *inference rules*. The core mission of CryptoVampire2 is to chain these rules to automate indistinguishability proofs. For Ex. 1, it is relevant to include Prop. 1[3].

Property 1 (PRF). *When* $\mathcal{H}(\_, \_)$ *behaves like a* Pseudo Random Function *[40], the following rule holds:*

$$\frac{u \sim v \vdash C[\mathsf{n}] \sim b \qquad m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} C[\_], m}{u \sim v \vdash C[\mathcal{H}(m, \mathsf{k})] \sim b}\, PRF$$

where k *and* n *are nonces,* $a \sim b$ *is the* indistinguishability predicate*, and* $m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} \_$ *is a* subterm relation *that ensures (i)* n *is* fresh*, (ii) the key* k *was not leaked nor misused, and (iii)* $\mathcal{H}(m, \mathsf{k})$ *is not used outside of the hole in* $C[\_]$*.*

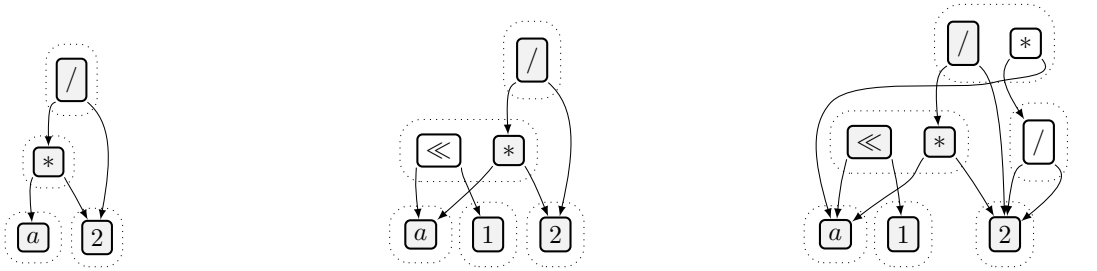
When these conditions hold, this rule allows the prover to replace $\mathcal{H}(m, \mathsf{k})$ *by a fresh nonce in an indistinguishability goal.*

A major contribution of this paper is to merge Prop. 1's (i), (ii) and (iii) into the single predicate $\_, \_, \_ \blacktriangleright_{PRF}$ ensuring each arguement search through the same term, leading to soundness modulo rewriting. Finally, properties not expressible as rewrite rules or inferences are given using first-order formulas. For example:

Theorem 1 (No Guessing). *For any nonce* $\mathsf{n} \in \mathcal{N}_{\mathrm{bc}}$*, BC term* t*,*

$$\mathsf{n} \blacktriangleright t \Rightarrow |\mathsf{n} \not\doteq t| \tag{6}$$

where $\mathsf{x} \blacktriangleright y$ *is also a subterm relation meaning that* x *is not used in* y*. That is, honest nonces cannot be guessed: a term that has no dependencies on a nonce (*$\mathsf{n} \blacktriangleright t$*) cannot be equal to it (*$|\mathsf{n} \not\doteq t|$*).*



(a) E-graph represents $(a * 2)/2$. **(b) Rewrite** $a * 2 \rightsquigarrow a \ll 1$. **(c) Rewrite** $(a * b)/c \rightsquigarrow a * (b/c)$.
Solid boxes denote e-nodes; dotted boxes denote e-classes; white e-classes are new e-classes.

Figure 1: Applying rewrites over an example e-graph (adapted from [67])

2.2 Proof Search modulo Equality

The presence of inference rules like Prop. 1's PRF naturally suggests Prolog-like reasoning: a depth-first search in the graph induced by the inference rules to find a proof tree, or in our case, a *Directed Acyclic Graph* (DAG). However, the requirement for robust equality reasoning—due to the many potentially complex equational rules—makes direct use of Prolog [26] impractical. The closest existing tool to our needs is egglog [68], an extension of Datalog with strong support for equality, but its logical fragment lacks the expressiveness required for our rules. Thus, we developed our own automated reasoning engine: golgge.

2.2.1 E-Graphs. Both egglog and golgge rely on *e-graphs* [60], efficient data structures for storing terms and a congruence relation (i.e., an equivalence relation $\approx$ such that $\vec{x} \approx \vec{y} \Rightarrow f(\vec{x}) \approx f(\vec{y})$). An e-graph is a collection of *e-classes*, each composed of *e-nodes*. An e-class represents the subset of an equivalence class stored in the e-graph, while an e-node describes one way to construct an element of this e-class by applying a function symbol to other e-classes. Typically, the relation $\approx$ is approximated using *oriented equation rules*. Fig. 1 shows an evolving e-graph under rewriting of $(a * 2)/2$ using the rules $x * 2 \rightsquigarrow y \ll 1$ and $(x * y)/z \rightsquigarrow x * (y/z)$.

2.2.2 Backtracking. egglog's key insight is to store facts as *e-classes* (subsets of equivalence classes in an e-graph). golgge reuses this insight but flips egglog's reasoning[4] from forward (building the DAG from axiom e-classes to the goal e-class, Datalog-style) to backward (building from the goal e-class toward the axiom e-classes, Prolog-style). golgge goes further than backward chaining, featuring powerful caching akin to SLG resolution [23].

Example 7. *Let a and b be two constants (i.e., honest functions with no arguments). To prove $\mathsf{n} \blacktriangleright f(a, \pi_1(\langle b, \mathsf{n}\rangle))$, golgge builds the proof tree* (7) *transparently using Ex. 6's rewriting rule* (4)*:*

$$\dfrac{\dfrac{}{\mathsf{n} \blacktriangleright \mathsf{c}_2}\, a \qquad \dfrac{}{\mathsf{n} \blacktriangleright \mathsf{c}_3}\, b}{\mathsf{n} \blacktriangleright \mathsf{c}_1}\, f \tag{7}$$

with $\mathsf{c}_1 :\approx f(a, \pi_1(\langle b, \mathsf{n}\rangle))$, $\mathsf{c}_2 :\approx a$, $\mathsf{c}_3 :\approx \pi_1(\langle b, \mathsf{n}\rangle) \approx b$.

This backward reasoning enables dynamic computation of applicable rules for each goal e-class, allowing arbitrarily complex rules. Notably, we can offload some goals to [45], translating them into first-order queries for external SMT solvers [12, 33, 51]. CryptoVampire2 coordinates Section 2.1 and Section 2.2 together. It translates the protocols and their constraints into a problem for golgge as described in Section 2.1. golgge then attempts to construct a proof, potentially delegating subgoals to [45]. If golgge succeeds, then the protocols are indeed indistinguishable (Th. 4).

[3] see its proof in the technical report [46]
[4] hence the name of the tool.

3 Preliminaries

We shall use vector notations $\vec{y}$ throughout as a shorthand for $y_1, \ldots, y_n$ when n is obvious from the context. We write $\mathrm{fv}(\phi)$ for the free variables of ϕ. We also write the *inference rule* $\frac{A_1 \ldots A_n}{B}$, with $n \geq 0$, to mean that $A_1 \wedge \ldots \wedge A_n \Rightarrow B$ is valid. All probability-related statements are defined in the technical report [46] .

3.1 Introduction to the CCSA—The BC Logic

CryptoVampire2's syntax and semantics extend the Bana-Comon Logic (BC Logic) [9]. This section briefly introduces to the BC Logic, while Section 3.2 introduces its extension in CryptoVampire2 and [45]. 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 the BC Logic are built from the following grammar:

$$t \coloneqq x \,|\, \mathsf{n} \,|\, f(\vec{t}) \,|\, g(\vec{t}) \tag{8}$$

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

A computational model $\mathbb{M}$ assigns elements of $\mathcal{N}_{\mathrm{bc}}$, $\mathcal{F}_{\mathrm{bc}}$, and $\mathcal{G}_{\mathrm{bc}}$ to polynomial-time Turing machines, defining an interpretation $[\![t]\!]^{\mathbb{M}}_{\mathrm{bc}}$ for each BC term t with respect to the security parameter η and a pair $\rho = (\rho_h, \rho_{\mathcal{A}})$ of random tapes (for honest randomness and the adversary, respectively):

(1) $[\![\mathsf{n}]\!]^{\mathbb{M}}_{\mathrm{bc}}(1^\eta, (\rho_h, \rho_{\mathcal{A}}))$, with $\mathsf{n} \in \mathcal{N}_{\mathrm{bc}}$, is a slice of length η of ρ_h, pairwise distinct for all elements of $\mathcal{N}_{\mathrm{bc}}$.
(2) For $f \in \mathcal{F}_{\mathrm{bc}}$,

$$[\![f(t_1, \ldots, t_n)]\!]^{\mathbb{M}}_{\mathrm{bc}}(1^\eta, \rho) = [\![f]\!]^{\mathbb{M}}\left(1^\eta, [\![t_1]\!]^{\mathbb{M}}_{\mathrm{bc}}(1^\eta, \rho), \ldots, [\![t_n]\!]^{\mathbb{M}}_{\mathrm{bc}}(1^\eta, \rho)\right) \tag{9}$$

where $[\![f]\!]^{\mathbb{M}}$ is a polynomial-time Turing machine with no direct access to ρ.

(3) For $g \in \mathcal{G}_{\text{bc}}$,

$$[\![g\,(t_1,\ldots,t_n)]\!]^{\mathbb{M}}_{\text{bc}}\,(1^\eta,\rho) = [\![g]\!]^{\mathbb{M}}\left(1^\eta,\rho_{\mathcal{A}},[\![t_1]\!]^{\mathbb{M}}_{\text{bc}}\,(1^\eta,\rho),\ldots,[\![t_n]\!]^{\mathbb{M}}_{\text{bc}}\,(1^\eta,\rho)\right), \quad (10)$$

where $\rho \coloneqq (\rho_h,\rho_{\mathcal{A}})$ and $[\![g]\!]^{\mathbb{M}}$ 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.

In this paper, we assume $\langle \_,\_\rangle$ (tuples), π_1, π_2 (projections), $\_ \doteq \_$ (equality), **If** _ **then** _ **else** _, `true`, and `false` are part of $\mathcal{F}_{\text{bc}}$. This lets us to define *boolean connectives*, which we distinguish from standard first-order boolean operators by overlining them (e.g., we write $a \barwedge b \coloneqq$ **If** a **then** b **else** `false` instead of $a \wedge b$).

Definition 1 (Indistinguishability). *Two BC terms a, b are* indistinguishable, *written $\mathbb{M} \vDash a \sim_{\text{bc}} b$, if for all polynomial-time attackers $\mathcal{A}$,*

$$\left|\begin{array}{l}\mathbb{P}\left(\mathcal{A}(1^\eta,\rho_{\mathcal{A}},[\![a]\!]^{\mathbb{M}}_{bc}\,(1^\eta,\rho))=1\right)\\ \quad -\mathbb{P}\left(\mathcal{A}(1^\eta,\rho_{\mathcal{A}},[\![b]\!]^{\mathbb{M}}_{bc}\,(1^\eta,\rho))=1\right)\end{array}\right| = \text{negl}\,(\eta) \quad (11)$$

where $\rho \coloneqq (\rho_h,\rho_{\mathcal{A}})$ with ρ_h and $\rho_{\mathcal{A}}$ independent random tapes. A function h is negligible *in η (written $\text{negl}\,(\eta)$) when $h(\eta) \overset{\eta\to\infty}{=} o\,(\eta^{-c})$ for all $c \in \mathbb{N}$.*

This strong semantic foundation allows us to prove BC axioms and rules using standard cryptographic tools. An important example is Th. 1 from Section 2 stating that it is hard to guess nonces:

Theorem (1 No Guessing). *For any nonce $\mathsf{n} \in \mathcal{N}_{\text{bc}}$, BC term t,*

$$\mathsf{n} \blacktriangleright t \Rightarrow |\mathsf{n} \bar{\neq} t| \quad (12)$$

where $\mathsf{x} \blacktriangleright y$ is also a subterm relation meaning that x is not used in y.

Proof Sketch. Assuming $\mathsf{n} \blacktriangleright t$, by induction on t, the random variables $[\![\mathsf{n}]\!]^{\mathbb{M}}_{\text{bc}}\,(1^\eta,\rho)$ and $[\![t]\!]^{\mathbb{M}}_{\text{bc}}\,(1^\eta,\rho)$ are independent. A detailed proof in the full logic is postponed to the technical report [46]. □

In practice, we restrict ourselves to a class $\mathbb{C}$ of models where function interpretations closely match their real-world counterparts (e.g., $\pi_1(\_)$ as a projection, $\mathcal{H}(\_,\_)$ is a PRF hashing function). The rules and formulas introduced in Section 2.1.4 describe this chosen class of models. We write $\mathbb{C} \vDash \varphi$ for validity in all models in $\mathbb{C}$.

3.2 Extensions to CryptoVampire2's Logic

3.2.1 The Notion of Protocols. The BC Logic effectively represents symbolic computations and can model a single protocol execution, where network inputs are modeled by applying an attacker function to the history of prior messages (the *trace*). However, protocol verification requires considering all possible executions. CryptoVampire2 adopts Squirrel's meta-logic approach, abstracting executions while formally unfolding back into the BC Logic for each interleaving. The result is the Symbolic Logic $\mathcal{S}(\mathcal{P},\mathcal{S},\mathcal{N},\mathcal{F},\mathcal{C})$ shown in Fig. 2.

Indices (variables $i, j, \ldots$ from $\mathcal{I}$) extend reasoning to unbounded protocols. **Timepoints** T represent when an atomic input/output

$$\begin{array}{rll}
I \coloneqq & i,j,\ldots & i,j \in \mathcal{I}, \text{ indices}\\
T \coloneqq & \tau & \text{timepoints}\\
& \mid \mathsf{a}\,[\vec{I}] & \mathsf{a}\,[\vec{_}] \in \mathcal{S}, \text{ step}\\
& \mid \mathsf{pred}\,(T) & \text{previous step}\\
A \coloneqq & T = T' \mid T < T' \mid T \le T' & \text{order over timepoints}\\
& \mid \mathsf{happens}\,(T) \mid N = N' & \\
t \coloneqq & u,v,\ldots \mid A & \\
& \mid \mathsf{n}\,[\vec{I}] & \mathsf{n}\,[\vec{_}] \in \mathcal{N}, \text{ nonces}\\
& \mid f\,[\vec{I}]\,(\vec{t}) & f\,[\vec{_}] \in \mathcal{F}, \text{ honest function}\\
& \mid \mathsf{att}(t) & \text{attacker function}\\
& \mid \textbf{Find } \vec{\alpha} \textbf{ st. } t \textbf{ then } t' \textbf{ else } t'' & \text{lookup}\\
& \mid \mathsf{M}@_P T & \text{macro}\\
P \coloneqq & p & p \in \mathcal{P}, \text{ protocol identifiers}\\
\mathsf{M} \coloneqq & \mathsf{input} \mid \mathsf{msg} \mid \mathsf{cond} \mid \mathsf{frame} \mid \mathsf{exec} \mid \mathsf{call}\,(\mathsf{c}\,[\vec{I}]) & \\
& & \mathsf{c}\,[\vec{I}] \in C, \text{ memory cell}
\end{array}$$

Terms built with this grammar form $\mathcal{S}(\mathcal{P},\mathcal{S},\mathcal{N},\mathcal{F},\mathcal{C})$, shortened to $\mathcal{S}$ when the parameters are obvious. $\vec{\alpha}$ is a vector of variables over timepoints and/or indices.

Figure 2: Grammar of the Symbolic Logic.

step is executed, either directly via an indexed identifier a or relatively via the predecessor operator pred. **Predicates** A reason about timepoint ordering ($=, <, \le$) or step execution (happens). **Terms** t follow the same structure to the BC Logic, with *indexed* nonce identifiers and function symbols from $\mathcal{N}$ and $\mathcal{F}$, respectively. We restrict ourselves to a single attacker function att. **Binders** (in the form of lookups) allow defining quantifiers:

$$\bar{\exists}\vec{\alpha}, t \coloneqq \textbf{Find } \vec{\alpha} \textbf{ st. } t \textbf{ then } \mathtt{true} \textbf{ else } \mathtt{false}, \quad (13)$$

$$\bar{\forall}\vec{\alpha}, t \coloneqq \bar{\neg}\bar{\exists}\vec{\alpha}, \bar{\neg} t. \quad (14)$$

Macros relate terms to executions or traces, taking a protocol identifier p and a timepoint T as arguments, representing input (input), output (msg), memory cell reads (call(_)), execution condition (cond), attacker knowledge (frame), and the conjunction of conditions of all executed steps (exec).

Definition 2 (Step). *A* step $\mathsf{a}\,[\vec{i}]$, *represented by (c,m,u), is an atomic input/output operation guarded by a condition:*

(1) c is the condition *guarding the step's execution (referred to by $\text{cond}_p\,(\mathsf{a}\,[\vec{i}])$).*

(2) m is the message *output if the condition succeeds (referred to by $\text{msg}_p\,(\mathsf{a}\,[\vec{i}])$).*

(3) $u\,(\mathsf{c}\,[\vec{j}])$ is the assignment *to cell $\mathsf{c}\,[\vec{j}] \in C$ (referred to by $\text{cell}_p\,(\mathsf{c}\,[\vec{j}],\mathsf{a}\,[\vec{i}])$)*

where p is the protocol and $\text{fv}\,(c,m) \subseteq \vec{i}$, $\text{fv}\,(u\,(\mathsf{c}\,[\vec{j}])) \subseteq \vec{i} \cup \vec{j}$.

We single out the step init, which always has no parameters and its condition set to `true`, as the first step executed in all protocols. Its output represents the attacker's initial knowledge.

Definition 3 (Protocol). *A* protocol p *is a tuple* $(\mathcal{S}, \preceq, \diamond)$ *composed of a set* $\mathcal{S}$ *of* steps *containing* init *and* constraints $\preceq$ *(partial order) and* $\diamond$ *(mutual exclusion) over* $\mathcal{S}$*. To ensure executability and representability, we require:*

1. *constraints and cell assignments are index-independent (e.g.,* $\mathsf{a}[\vec{i}] \preceq \mathsf{b}[\vec{j}] \Leftrightarrow \mathsf{a}[\vec{k}] \preceq \mathsf{b}[\vec{l}]$*);*
2. *step definitions only reference timepoints smaller than themselves according to* $\preceq$ *and protocol* p *and recursive cell writes statically terminate;*
3. init *is the smallest element of* $\mathcal{S}$ *according to* $\preceq$*.*

Remark 1 (Many init steps). *When attacker knowledge does not fit in a single step (e.g., publishing a whole sequence* $(pk)_{\vec{i}}$ *of public keys), we introduce extra initialization steps: unguarded steps smaller by* $\preceq$ *than all steps in the main body of the protocol.*

3.2.2 *Executions.*

We model protocol execution with a *trace*:

Definition 4 (Trace). *Given a protocol* $p = (\mathcal{S}, \preceq, \diamond)$*, a* trace $\mathbb{T}$ *is a tuple* $(\mathbb{I}, \mathbb{S}, \sigma_{\mathbb{I}}, \leq_{\mathbb{S}})$ *such that:*

1. $\mathbb{I}$ *is a* finite set *of indices;*
2. $\sigma_{\mathbb{I}} : \mathcal{I} \to \mathbb{I}$*, idempotent on* $\mathbb{I}$*, assigns index variables to* $\mathbb{I}$*;*
3. $\mathbb{S} \subseteq \{\mathsf{a}[\sigma_{\mathbb{I}}(i_1), \ldots, \sigma_{\mathbb{I}}(i_n)] | \mathsf{a}[i_1, \ldots, i_n] \in \mathcal{S}\}$*, is the downward closed*[5] *set by* $\preceq$ *of all* executed steps*;*
4. $\leq_{\mathbb{S}}$ *is a* total order *over* $\mathbb{S}$ compatible *with* $\preceq$ *and* $\diamond$*:*

$$\mathsf{a}[\vec{i}] \preceq \mathsf{b}[\vec{j}] \Rightarrow \mathsf{a}[\vec{i}] \leq_{\mathbb{S}} \mathsf{b}[\vec{j}], \tag{15}$$

$$\mathsf{a}[\vec{i}] \diamond \mathsf{b}[\vec{j}] \Rightarrow \mathsf{a}[\vec{i}] <_{\mathbb{S}} \mathsf{b}[\vec{j}] \wedge \mathsf{b}[\vec{j}] <_{\mathbb{S}} \mathsf{a}[\vec{i}]. \tag{16}$$

(16) prevents two mutually exclusive steps from being in $\mathbb{S}$ simultaneously, and (15) and downward closure ensures init is in $\mathbb{S}$ and is its smallest element.

Example 8. *Reusing Ex. 1, assume two tags and two rounds. The protocol splits into steps* init, $\mathsf{T}[i, j]$, $\mathsf{Rs}[i, j]$*, and* $\mathsf{Rf}[i]$ *as in Ex. 4, with constraints from Equation 3. Let* $\mathcal{I} := \mathbb{N}$ *and consider:*

$$\mathcal{S}_1 = \{\mathsf{init} <_1 \mathsf{T}[1,2] <_1 \mathsf{Rs}[1,2]\}, \tag{17}$$

$$\mathcal{S}_2 = \{\mathsf{init} <_2 \mathsf{T}[1,1] <_2 \mathsf{Rs}[1,2] <_2 \mathsf{T}[2,2]\}, \tag{18}$$

$$\mathcal{S}_3 = \{\mathsf{init} <_3 \mathsf{Rf}[1] <_3 \mathsf{T}[1,1] <_3 \mathsf{Rs}[1,1]\}. \tag{19}$$

(17) *describes a valid trace with* $\mathbb{I} = \{1, 2\}$*, as is* (18) *despite step* $\mathsf{Rs}[1,2]$ *failing. However,* (19) *is not a valid trace as it contradicts* $\diamond$ *despite the condition of all the steps seemingly holding.*

We restrict ourselves to protocols sharing some overall structure enforcing lock step execution by taking inspiration from the notion of *Bi-Process* [19, 24] used in other automated protocol verifiers supporting indistinguishability like ProVerif [17].

Definition 5 (Compatible Protocols). *Two protocols are* compatible *if they share step identifiers and constraints.*

Throughout the remainder of this paper, we will assume $\mathcal{P}$ consists of pairwise compatible protocols.

Example 9. *In Ex. 1,* (1) *and* (2) *are compatible.*

Definition 6 (Unfolding). *Given a trace* $\mathbb{T}$ *and a term* t *from the Symbolic Logic,* $(t)^{\mathbb{T}}$ *is its* unfolding *into a BC term (the technical report [46]).*

[5] i.e., such that if $y \preceq x$ and $x \in \mathbb{S}$ then $y \in \mathbb{S}$.

Example 10. *Continuing Ex. 1, let us unfold* $\mathsf{input}@_{(1)}(\mathsf{Rs}[1,2])$ *according to trace* $\mathbb{T}_{(17)}$ *described by Ex. 8's* (17)*:*

$$\left(\mathsf{input}@_{(1)}(\mathsf{Rs}[1,2])\right)^{\mathbb{T}_{(17)}} := \mathsf{att}\left(\begin{array}{l}\langle \left(\mathrm{cond}_{(1)}(\mathsf{T}[1,2])\right)^{\mathbb{T}_{(17)}} \bar{\wedge} \left(\mathrm{cond}_{(1)}(\mathsf{init})\right)^{\mathbb{T}_{(17)}}; \\ \quad \langle \textbf{If } \left(\mathrm{cond}_{(1)}(\mathsf{T}[1,2])\right)^{\mathbb{T}_{(17)}} \bar{\wedge} \left(\mathrm{cond}_{(1)}(\mathsf{init})\right)^{\mathbb{T}_{(17)}} \\ \qquad \textbf{then } \left(\mathrm{msg}_{(1)}(\mathsf{T}[1,2])\right)^{\mathbb{T}_{(17)}} \textbf{ else } \mathsf{fail}; \\ \qquad\qquad \left(\mathrm{msg}_{(1)}(\mathsf{init})\right)^{\mathbb{T}_{(17)}} \rangle\rangle\end{array}\right) \tag{20}$$

We extend indistinguishability to the Symbolic Logic: Given a computational model $\mathbb{M}$ and a trace $\mathbb{T}$, we write $\mathbb{M}, \mathbb{T} \vDash a \sim b$ when $\mathbb{M} \vDash (a)^{\mathbb{T}} \sim_{\mathrm{bc}} (b)^{\mathbb{T}}$. We abbreviate $\mathbb{M}, \mathbb{T} \vDash a \sim b$ or $\mathbb{C}, \mathbb{T} \vDash a \sim b$ as $a \sim b$ when context permits. We write $\vdash_{\mathbb{P}} a$ to denote $a \sim \mathsf{true}$ (a *trace property*). Throughout this paper, we use $\mathbb{C}, \mathbb{T} \vDash\!\vdash_{\mathbb{P}} a \doteq b$ as our congruence relation, written $a \equiv b$.

Definition 7 (Indistinguishable Protocols). *Two* compatible *protocols* p_1 *and* p_2 *are* observationally equivalent *or* indistinguishable *in a class of models* $\mathbb{C}$ *if for all timepoints* τ *and traces* $\mathbb{T}$*,*

$$\mathbb{C}, \mathbb{T} \vDash \mathsf{frame}@_{p_1}\tau \sim \mathsf{frame}@_{p_2}\tau. \tag{21}$$

Notations 1. *We write* $[\![t]\!]^{\eta}_{\mathbb{M},\mathbb{T}}(\rho)$ *for* $[\![(t)^{\mathbb{T}}]\!]^{\mathbb{M}}_{bc}(1^{\eta}, \rho)$*, omitting* $\mathbb{M}$ *and/or* $\mathbb{T}$ *when obvious.*

We link *trace properties* to the notion of overwhelming sequences (Def. 8) by noting that $\vdash_{\mathbb{P}} P$ means that $\left([\![P]\!]^{\eta}_{\mathbb{M},\mathbb{T}}(\rho) = 1\right)_{\eta \in \mathbb{N}}$ is overwhelming.

Definition 8 (Overwhelming). *A sequence of sets or events* $(E_\eta)_{\eta \in \mathbb{N}}$ *is* overwhelming *if* $|1 - \mathbb{P}(E_\eta)| = \mathrm{negl}(\eta)$.

Thus far, all notions introduced map directly to those of Squirrel [5], with the sole exception that macros are tagged with a protocol identifier. This correspondence immediately yields Prop. 2.

Property 2 (Intercompatibility). *Two compatible and executable protocols are indistinguishable in CryptoVampire2 if and only if they are indistinguishable in Squirrel [5, 6].*

Remark 2. *Note that Prop. 2 only considers the original Squirrel [5, 6] and does not account for some of its later extensions such as high-order [29] or quantum soundness [28].*

CryptoVampire2 mainly considers the Symbolic Logic. Notably, our use of golgge is constrained to it. However, we sometimes requires more precision than golgge can achieve (e.g., for some trace properties), prompting offloads to [45] which uses a further extension to FOL.

3.2.3 *As a First-Order Logic.*

As noted elsewhere [4, 9, 45][6], the BC Logic's semantics do not match classical FOL.

Example 11 (Counterexample to Classical Semantics). *Consider* `first_bit(n)`*, with* n *a nonce. It can be distinguished from both* `true` *and* `false`*.*

Nonetheless, as shown in [45], we can extend the Symbolic Logic to be compatible with classical FOL.

[6] in the technical report of [9]

Definition 9 (Evaluated Logic). *We define the* Evaluated Logic $\mathcal{L}(\mathcal{P}, \mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{C})$ *as:*

$$\begin{aligned} t &\coloneqq \ldots \quad \text{as in Fig. 2,} \\ \phi &\coloneqq |t| \mid P(\vec{t}) \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \neg\phi \mid \exists \vec{x}.\phi \mid \forall \vec{x}.\phi, \end{aligned} \tag{22}$$

where P denotes predicates (e.g., other subterm relations).

Given a cryptographic model $\mathbb{M}$ and a trace $\mathbb{T}$, we link the Evaluated Logic with the Symbolic Logic—and consequently the rest of the CCSA literature—using models $\mathcal{L}^{\mathbb{M},\mathbb{T}}$. These Tarskian models [66] are restricted to be a Herbrand model [42] over $\mathcal{S}$ and evaluate $|t|$ to $[\![t]\!]^{\eta}_{\mathbb{M},\mathbb{T}}(\rho)$:

$$\mathcal{L}^{\mathbb{M},\mathbb{T}}(t)\,(1^{\eta}, \rho) \coloneqq t \quad \text{when } t \in \mathcal{S}(\mathcal{P}, \mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{C})$$

$$\mathcal{L}^{\mathbb{M},\mathbb{T}}(|t|)\,(1^{\eta}, \rho) \coloneqq \begin{cases} \mathbf{0} & \text{if } [\![t]\!]^{\eta}_{\mathbb{M},\mathbb{T}}(\rho) = 0 \\ \mathbf{1} & \text{otherwise} \end{cases}$$

Property 3 (Relation Between Logics).

$$\mathbb{P}\left(\mathcal{L}^{\mathbb{M},\mathbb{T}}(|t|)\,(1^{\eta}, \rho) = \mathbf{0}\right) = \mathrm{negl}(\eta) \;\; \textit{iff. } \mathbb{M}, \mathbb{T} \vDash_{\mathbb{P}} t. \tag{23}$$

for any $t \in \mathcal{S}(\mathcal{P}, \mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{C})$.

Theorem 2 (Weak First-Order Compatibility). *Let φ, ϕ be quantifier-free formulas of $\mathcal{L}(\mathcal{P}, \mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{C})$ such that:*

(1) $\vec{x} \coloneqq \mathrm{fv}(\varphi)$ and $\vec{y} \coloneqq \mathrm{fv}(\phi)$;
(2) for all $\vec{t}, \mathbb{C}, \mathbb{T} \vDash_{\mathbb{P}} \varphi\{\vec{x} \mapsto \vec{t}\}$;
(3) $(\forall \vec{x}.\varphi) \Rightarrow (\forall \vec{y}.\phi)$ is valid *in FOL.*

Then $\mathbb{C}, \mathbb{T} \vDash_{\mathbb{P}} \phi\{\vec{y} \mapsto \vec{t}\}$ for all $\vec{t}$.

Prop. 3 and Th. 2 enable offloading trace properties to classical automated theorem provers for FOL. This is the basis of [45], which we use when the e-graph approach is not complete enough. As the topic is quite orthogonal to indistinguishability, we refer the reader to [45] for more details on the FOL translation specifically and a stronger version of Th. 2.

4 GOLGGE

Automating indistinguishability proofs requires tackling two significant challenges:

(A) Even the simple Ex. 1 requires a wide range of equality rules, from basic semantics (e.g., Ex. 6) to advanced properties like authentication (e.g., Lem. 1) and macro unfolding. While Ex. 6 reduces term size, macro unfolding increases it, and Lem. 1 transforms terms into structurally-unrelated forms. Despite existing research [49], we were not able to complete the set of equations.

(B) The rules of the CCSA calculus can be highly complex: for instance, some encode inductive reasoning (e.g., induction on traces), others require arbitrary trace properties (e.g., step ordering), and some introduce fresh symbols.

This combination of challenges means no existing tool fully meet our needs, necessitating the development of GOLGGE. The closest existing tool to our needs is EGGLOG [68], a powerful equality saturation engine and Datalog theorem prover. Like classic Datalog [22], it derives all facts from initial facts using inference rules, but unlike Datalog, it supports arbitrary rewriting. EGGLOG uses e-graphs [60] to efficiently represent equivalence classes under the congruence relation induced by rewriting. Thus, facts are no longer simple terms, but full equivalence classes instead. However, EGGLOG's Datalog heritage presents limitations for CryptoVampire2: it requires pre-enumerating all base facts, which is impractical for our calculus where facts like trace properties and introduction of fresh terms emerge dynamically during proofs. Nonetheless, EGGLOG serves as a strong foundation for GOLGGE, particularly its e-graph infrastructure, which we adapt by reversing the reasoning direction from forward to backward chaining to better handle the unbounded nature of our proof obligations.

This section describes details of GOLGGE. After more formally introducing e-graphs in Section 4.1, we present a simplified version of the engine in Section 4.2 and discuss extensions and optimizations.

4.1 E-Graphs and Equality Saturation

Definition 10 (E-Graph). *An* e-graph $\mathfrak{G}$ *over a signature* Σ *and a congruence relation* $\approx$ *consists of* e-nodes *and* e-classes*:*

(1) An e-class $\mathrm{a} \coloneqq \{\mathrm{n}_1, \ldots, \mathrm{n}_m\}$ *is a set of e-nodes* n_i*, each representing a subset of an equivalence class stored in* $\mathfrak{G}$*.*
(2) An e-node n *represents a way to construct its e-class* b*. It takes the form* $f(\vec{\mathrm{a}})$ *with* $f \in \Sigma$ *and* $\vec{\mathrm{a}}$ *a vector of e-classes, expressing that for all term vectors* $\vec{x}, \vec{y} \in \vec{\mathrm{a}}$*,* $f(\vec{x}) \approx f(\vec{y})$ *and both belong to* b*.*

Fig. 1 illustrates several e-graphs. E-graphs are widely used in automated theorem proving [32, 33, 61] and form a powerful basis for equality saturation engines like EGG [67]. For the latter, the relation $\approx$ is typically approximated by *directed equality rules*.

Definition 11 (Directed Equality Rules). *A* directed equality rule *has the form*

$$x_1 = t_1, \ldots, x_n = t_n \rightsquigarrow y_1 = t'_1, \ldots, y_m = t'_m \tag{24}$$

where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are variables and $t_1, \ldots, t_n, t'_1, \ldots, t'_m$ are patterns *(i.e., terms with variables) such that* $\mathrm{fv}(t'_1, \ldots, t'_m) \cup \{y_1, \ldots, y_m\} \subseteq \mathrm{fv}(t_1, \ldots, t_n) \cup \{x_1, \ldots, x_n\}$.

A rule is *sound* when for any substitution σ, if $\sigma(x_i) \approx t_i\sigma$ for all i, then $\sigma(y_j) \approx t'_j\sigma$ for all j. We simply write $t \rightsquigarrow t'$ for $x = t \rightsquigarrow x = t'$ with $x \notin \mathrm{fv}(t, t')$. This aligns with the intuitive notion of equality rules.

The orientation of rules helps equality saturation solvers avoid infinite loops by only adding terms in one direction. Rewriting rules underapproximate (often non-tightly) the congruence relation. Given an e-graph $\mathfrak{G}$, we write $\equiv_{\mathfrak{G}}$ for the induced congruence relation, and, assuming sound equality rules, we get:

Property 4 (E-Graphs Underapproximate). *For $t, t' \in \mathfrak{G}$,*

$$t \equiv_{\mathfrak{G}} t' \Rightarrow t \approx t'. \tag{25}$$

Definition 12 (Saturation). *An e-graph is* saturated *when it is stable under all its equality rules.*

In particular, once $\mathfrak{G}$ is saturated, $\equiv_{\mathfrak{G}}$ is a proper congruence relation. Fig. 1 exemplifies the saturation process.

4.2 Base Decision Procedure

GOLGGE expects the following inputs: (i) a set $\mathcal{R}$ of *directed equality rules* (Def. 11), (ii) a set $\mathcal{T}$ of *inference rules* (Def. 13), (iii) a term g to serve as a *goal* to prove.

Definition 13 (Inference Rules). *Given patterns $Q, R_1, \ldots, R_n$ such that $\mathrm{fv}(R_1, \ldots, R_n) \subseteq \mathrm{fv}(Q)$, the rule*

$$\frac{R_1 \quad \ldots \quad R_n}{Q} \tag{26}$$

is sound modulo $\approx$ if for all ground propositions $R'_1, \ldots, R'_n, Q'$ where $R'_1, \ldots, R'_n$ hold, if we can find a substitution σ such that $R_i\sigma \approx R'_i$ and $Q\sigma \approx Q'$, then Q holds as well.

Remark 3. *The constraints on free variables in a GOLGGE inference rule are the reverse of those in EGGLOG.*

The tool attempts to build a proof tree for g using the rules in $\mathcal{T}$ and $\mathcal{R}$. Facts, such as the goal, are represented by their e-class (e.g., g for g) in an EGG e-graph continuously saturated using $\mathcal{R}$. Simultaneously, GOLGGE backtracks over the rules in $\mathcal{T}$, searching for a rule whose conclusion unifies with g by *e-matching* and then attempts to prove each premise. If it fails, it backtracks. Algorithm 1 shows a simplified implementation of GOLGGE.

Algorithm 1 Simplified version of GOLGGE

```
function PROVE_1 (g)
saturate (R)
for (R_1 ... R_n)/Q ∈ T do
  if Qσ ≈ g then
    R_1, ..., R_n ← add (R_1σ, ..., R_nσ)
    if ∀i, PROVE_1 (R_iσ) then
      return true
return false
end function
```

where `saturate` ($\mathcal{R}$) saturates the underlying e-graph with the rules in $\mathcal{R}$, σ is a substitution found by e-matching, and `add` adds terms to the e-graph, returning the potentially new e-class.

In the end, the proof generated by the execution of GOLGGE is a proof tree of $\mathcal{T}$, where substitutions by $\equiv_{\mathfrak{G}}$ are interleaved between each rule application. This means all predicates must *at least* be compatible with $\approx$ (i.e., if $\vec{x} \approx \vec{y}$ then $P(\vec{x}) \approx P(\vec{y})$) and potentially more if a predicate can be rewritten into another. Another notable point is that $\mathfrak{G}$ only grows during execution. Consequently, $\equiv_{\mathfrak{G}}$ also grows monotonically, and no backtracking over it may occur. This property significantly reduces complexity during backtracking. It is a key differentiating factor between GOLGGE and EGGLOG's approach to e-graphs and a typical SMT solver's treatment [33].

Theorem 3 (Soundness of GOLGGE). *If all rules in $\mathcal{T}$ and all equality rules are sound and PROVE$_1$(g) returns* `true`*, then for all predicates $P \in$ g, P holds.*

Proof. By induction on Algorithm 1. □

4.3 Extensions and Optimizations

4.3.1 Dynamic Inference Rules. Within CryptoVampire2, restricting ourselves to a predetermined finite set of inference rules is insufficient. For example, the PRF rule of Prop. 1 uses *subterm relations*, requiring proving that some terms are distinct (i.e., $\vdash_{\mathbb{P}} a \neq b$). These queries are impossible for the e-graph, as it only underapproximates equality (Prop. 4). Thus, a natural extension is to determine proof obligations dynamically as we pattern-match on the goal. This allows us to: (i) apply optimized decision procedures for some goals (e.g., subterm reasoning), (ii) trigger transformations on the e-graph that would otherwise be complex (e.g., substitutions in contexts), and (iii) offload queries to [45] (see Ex. 12).

Example 12 (Trace Queries). *With dynamic rules, the goal $\vdash_{\mathbb{P}} \varphi$ can be offloaded to [45] and treated as an axiom upon success; otherwise, it behaves as if no rule applies.*

Dynamic rules are more expensive than regular ones, especially offloads to the trace solver, which must often timeout to fail. This makes the caching of Section 4.3.2 yet more critical.

Remark 4 (Evolution of $\equiv_{\mathfrak{G}}$). *While dynamic rules can modify the e-graph directly (e.g., add terms, merge e-classes), they* must not *introduce modifications that contradict Prop. 4, as GOLGGE does not backtrack on $\equiv_{\mathfrak{G}}$.*

In this new setting, we consider $\mathcal{T}$ as a *function* from e-classes to sets of sets of e-classes representing the next proof obligations.

Example 13. *$\mathcal{T}$ (x) $= \{\emptyset\}$ expresses that* x *is an* axiom*, while $\mathcal{T}$ (x) $= \emptyset$ indicates it is unprovable.*

In practice, such sets are lists whose order can be relied on for stateful operations and optimizations (e.g., executing cheap pruning goals before expensive ones).

4.3.2 Loop Prevention and Memoization. Loops can arise quickly during the search, as shown in Ex. 14.

Example 14 (Loop). *Recall* (4) *from Ex. 6 and the PRF rule of Prop. 1. Consider $u \sim v \vdash \pi_1(\langle x, \mathcal{H}(y, \mathrm{k})\rangle) \sim b$. We obtain:*

$$\dfrac{\dfrac{u \sim v \vdash \pi_1(\langle x, \mathcal{H}(y, \mathrm{k})\rangle) \sim b \qquad by\ (4)}{u \sim v \vdash \pi_1(\langle x, \mathrm{n}\rangle) \sim b}\,\textsc{Subst} \qquad \ldots}{u \sim v \vdash \pi_1(\langle x, \mathcal{H}(y, \mathrm{k})\rangle) \sim b}\,\textit{PRF} \tag{27}$$

thus creating a loop.

To mitigate this, we use *memoization*, tracking currently proven goals and pruning branches that reuse them. This complements *failure and success caching* similar to Prolog's SLG resolution [23]. Unlike in Prolog, we cache whole e-classes, significantly reducing the search space, as seen in Ex. 15.

Example 15 (Caching). *Following Ex. 14, caching $u \sim v \vdash x \sim b$ also caches $u \sim v \vdash \pi_1(\langle x, \mathcal{H}(y, \mathrm{k})\rangle) \sim b$, skipping the expensive PRF rule. Caching also lets GOLGGE to detect the loop in Ex. 14.*

Simple cases like Ex. 14 effectively make memoization a requirement for termination.

With all optimizations, GOLGGE behaves as in Algorithm 2, where mem is the memoization hashmap.

This implementation of memoization is not tight. It is nonetheless sound (Prop. 5) and empirically sufficient for protocol verification (Section 6).

Property 5. *If PROVE$_2$(g) returns* `true`*, then so does PROVE$_1$(g).*

Proof. By induction in Algorithm 2 on Prop. 5 and mem [g] = `true` $\Rightarrow$ PROVE$_2$(g) = `true`. □

Algorithm 2 GOLGGE

```
function PROVE_2 (g)
saturate (R)
if g ∈ mem then
  return mem [g]
mem [g] ← false
for {R_1, ..., R_n} ∈ T (g) do
  if ∀i, PROVE_2 (R_i) then
    mem [g] ← true
    return true
return false
end function
```

$\mathcal{T}$ can mutate the e-graph and is responsible for adding proof obligations to it.

5 From Theory to Practice: CryptoVampire2 for Indistinguishability Goals

In this section, we integrate all previously introduced concepts and instantiate them in the form of CryptoVampire2's indistinguishability solver.

5.1 General Shape of the Proofs

Let p_1 and p_2 be two compatible protocols using steps from $\mathcal{S}$, and $\mathbb{T}$ a trace. Recall from Def. 7 that the goal of CryptoVampire2 is to prove:

$$\mathsf{frame}@p_1\,\tau \sim \mathsf{frame}@p_2\,\tau \tag{21}$$

This typically requires induction on τ using $\mathbb{T}$'s $\leq_{\mathbb{S}}$ ordering. Induction is notoriously difficult [21, 41], especially in determining when to initiate it. Nonetheless, CryptoVampire2 *always* first applies induction (via the Induction rule of Fig. 3) and adapts the standard closing procedure to address potential missing assumptions of the induction hypothesis (see Section 5.2.4).

This culminates in CryptoVampire2 running GOLGGE once per step, backtracking over the rules definied in this paper, in order to prove the inductive case. Whenever soundness or performance requires it, GOLGGE offloads to [45]'s SMT solving.

5.2 New and Improved Decision Procedures

Throughout this section, we use Prop. 1's PRF as our leading example. Other cryptographic notions typically behave similarly. We remind the reader of the rule:

Property (1). *When $\mathcal{H}(\_,\_)$ behaves like a* Pseudo Random Function *[40], the following rule holds:*

$$\frac{\mathsf{frame}@p_1\,\mathsf{pred}(\mathsf{a}) \sim \mathsf{frame}@p_2\,\mathsf{pred}(\mathsf{a}) \vdash \mathsf{frame}@p_1\,\mathsf{a} \sim \mathsf{frame}@p_2\,\mathsf{a} \quad \text{for all } \mathsf{a} \in \mathcal{S}\setminus\{\mathsf{init}\} \qquad \mathsf{true} \sim \mathsf{true} \vdash \mathsf{frame}@p_1\,\mathsf{init} \sim \mathsf{frame}@p_2\,\mathsf{init}}{\mathsf{frame}@p_1\,\tau \sim \mathsf{frame}@p_2\,\tau}\,\textsc{Induction}$$

Figure 3: Induction rule

$$\frac{u \sim v \vdash C\,[\mathsf{n}] \sim b \qquad m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} C\,[\_], m}{u \sim v \vdash C\,[\mathcal{H}(m,\mathsf{k})] \sim b}\,PRF$$

with the notations introduced in Section 2's Prop. 1.

In this section, we explore the rule sets designed to utilize rules like PRF. We introduce how to efficiently search for the context $C\,[\_]$ (Section 5.2.1), how to compute $C\,[\mathsf{n}]$ from $C\,[\mathcal{H}(m,\mathsf{k})]$ (Section 5.2.2), and how to solve for $m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF}$ (Section 5.2.3). Finally, we extend the Deduce rules [6] (Section 5.2.4) to close goals.

5.2.1 Looking for Candidates. Searching for surface-level patterns in e-classes is a solved problem addressed via e-matching [32]. However, rules such as PRF require objects of the form $C\,[\mathsf{f}(\vec{x})]$, an e-class containing a pattern in one of its children. To find contexts of the form $C\,[m]$ with $\vec{x} := \mathrm{fv}(m)$, we introduce a function marker. This function interacts with the e-graph using rules:

$$m \rightsquigarrow \mathsf{marker}(\vec{x}, m) \tag{28}$$

$$\mathsf{f}(\vec{u}, \mathsf{marker}(\vec{z}, y), \vec{v}) \rightsquigarrow \mathsf{marker}(\vec{z}, \mathsf{f}(\vec{u}, y, \vec{v})) \tag{29}$$

where f is a constructor of $\mathcal{S}$. We have Prop. 6 by induction.

Property 6 (Marker). *In a saturated e-graph $\mathfrak{G}$, $\mathsf{marker}(\vec{x}, t) \equiv_{\mathfrak{G}} t$ and t is of the form $C\,[m]$.*

Such markers allow us to trigger rules efficiently, keeping them simple. Overall, Ex. 16 instantiates this concept for PRF.

Example 16 (PRF Marker). *We build $\mathsf{candidate}_{PRF}(m, \mathsf{k}, t)$ as a marker for the PRF rule to find contexts of the form $C\,[\mathcal{H}(m,\mathsf{k})]$. We then have the following rewriting rules:*

$$\mathcal{H}(m,\mathsf{k}) \rightsquigarrow \mathsf{candidate}_{PRF}(m, \mathsf{k}, \mathcal{H}(m,\mathsf{k})) \tag{30}$$

$$\mathsf{f}(\vec{x}, \mathsf{candidate}_{PRF}(m,\mathsf{k},y), \vec{z}) \rightsquigarrow \mathsf{candidate}_{PRF}(m,\mathsf{k},\mathsf{f}(\vec{x},y,\vec{z})) \tag{31}$$

where k is a nonce and f is a constructor of $\mathcal{S}$.

Now, we can simply trigger PRF on:

$$u \sim v \vdash \mathsf{candidate}_{PRF}(m,\mathsf{k},a) \sim b \tag{32}$$

PRF still requires transforming $C\,[\mathcal{H}(m,\mathsf{k})]$ into $C\,[\mathsf{n}]$.

5.2.2 Context and Substitutions over General Terms. Many cryptographic rules feature terms of the form $C\,[\_]$. Such a *context* C represents "a term with holes", encoding substitution of arbitrary terms. For instance, in PRF, it is used to convey that one can replace all instances of $\mathcal{H}(m,\mathsf{k})$ by n. However, such a notion is not well-defined for e-classes as shown in Ex. 17.

Example 17. *Consider $C\,[\mathcal{H}(m,\mathsf{k})] \equiv \mathcal{H}(m,\mathsf{k}) \equiv$ **If** $\mathsf{n} = \mathcal{H}(m,\mathsf{k})$ **then** 0 **else** $\mathcal{H}(m,\mathsf{k})$. What should $C\,[\mathsf{n}]$ be?*

(1) Taking $C\,[\_] := \_$ yields $C\,[\mathsf{n}] := \mathsf{n}$

*(2) Taking $C\,[\_] :=$ (**If** $\mathsf{n} = \_$ **then** 0 **else** $\_$) yields $C\,[\mathsf{n}] := 0$*

Already those two options point to two different e-classes for $C\,[\mathsf{n}]$. Moreover, (1) can be used for PRF but subterm checks reject (2).

The difficulty comes from choosing a C simply from what the markers of Section 5.2.1 highlight. This choice affects the related proofs obligations, notably the subterm checks. Flipping the problem around, the proofs of subterms effectively choose a C for us. Thankfully, GOLGGE's memoization effectively saves the proofs of

$$\textsc{PrfInst}\ \frac{}{m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} \mathcal{H}(m, \mathsf{k})} \qquad \textsc{PrfNonce}\ \frac{\vdash_{\mathbb{P}} \mathsf{k} \neq \mathsf{n}' \qquad \vdash_{\mathbb{P}} \mathsf{n} \neq \mathsf{n}'}{m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} \mathsf{n}'}$$

$$\textsc{PrfNeqM}\ \frac{\vdash_{\mathbb{P}} m \neq m' \qquad m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} m'}{m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} \mathcal{H}(m', \mathsf{k})}$$

$$\textsc{PrfFa}\ \frac{m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} t_1 \qquad \dots \qquad m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} t_n \qquad f \in \mathcal{F} \setminus \{\mathcal{H}\}}{m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} f(t_1, \dots, t_n)}$$

$$\textsc{PrfInput}\ \frac{\textsc{VampPrf}_{\mathsf{m},p}(m, \mathsf{k}, T) \qquad \mathsf{m} \in \{\mathsf{exec}, \mathsf{frame}, \mathsf{cell}(\_)\}}{m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} \mathsf{m}@_p\, \mathsf{pred}(T)}$$

where n is a nonce and $\textsc{VampPrf}_{\mathsf{m},p}(m, \mathsf{k}, \_)$ is a built-in decision procedure calling [45].

Figure 4: Relevant simplified subset of $m, \mathsf{k}, \_ \blacktriangleright_{PRF}$ rules

each predicate that is proven; we can use such proofs to build the new term. Gathering everything together, the rule PRF becomes:

$$\frac{\begin{array}{c} u \sim v \vdash \mathsf{subst}_{PRF}(m, \mathsf{k}, a, \mathsf{n}; (m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} a)) \sim b \\ m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} a, m \end{array}}{u \sim v \vdash \mathsf{candidate}_{PRF}(m, \mathsf{k}, a) \sim b}\ \text{PRF}$$

A relevant subset of the rules for $m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} t$ is shown in Fig. 4 giving an intuition on how to construct a $C[\_]$ from a proof.

5.2.3 Subterms and Equality. In the CCSA, cryptography-related axioms often translate *semantic* observations (e.g., $a \sim b$ or $\vdash_{\mathbb{P}} a \doteq b$) into *syntactic* facts, typically involving subterm searches [5, 45]. These two modes of reasoning often interact poorly, leading to inconsistencies if not handled carefully, especially with CCSA equality [45]. Here, we avoid such syntactic searches and replace them with other predicates. We remark that these subterm searches implicitly encode proofs of existence for polynomial-time algorithms—potentially with access to specific oracles—that an attacker could use to break an underlying cryptographic assumption. This often comes hand in hand with Section 5.2.2's context retrieval.

Example 18 (Subterms as Proof). *In PRF, $m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} t$ means that $[\![t]\!]^{\eta}_{\mathbb{M},\mathbb{T}}$ is a PTIME algorithm computing t's value using a hashing oracle but not k nor n (i.e., with no direct access to the corresponding slices of ρ_h outside of negligible events). Thus PRF's condition turns a distinguisher between $C[\mathsf{n}]$ and $C[\mathcal{H}(m, \mathsf{k})]$ into a valid attacker in a PRF sub-security game. Similarly, both PRF and Th. 1, $\mathsf{n} \blacktriangleright t$ proves that $[\![t]\!]^{\eta}_{\mathbb{M},\mathbb{T}}$ is a PTIME algorithm that does not use n. Refer to the technical report [46] for formal statements for each rule.*

In Fig. 4, PrfNonce, PrfNeqM, and PrfFa already provide a good intuition for Ex. 18. These rules sometimes produce goals of the form $\vdash_{\mathbb{P}} a \neq b$. This stems from Prop. 4's underapproximation: we need to call [45] to prove overwhelming disequality of terms as we cannot rely on the incomplete $\equiv_{\mathfrak{G}}$. Intuitively, this ensures $[\![t]\!]^{\eta}_{\mathbb{M},\mathbb{T}}$ never uses forbidden terms with non-negligible probability.

$$\textsc{DeduceFa}\ \frac{u, v \rhd_{\#} x_1, y_1 | h_1, h_2 \qquad \dots \qquad u, v \rhd_{\#} x_n, y_n | h_1, h_2}{u, v \rhd_{\#} f(x_1, \dots, x_n), f(y_1, \dots, y_n) | h_1, h_2}$$

$$\textsc{DeduceIte}\ \frac{\begin{array}{c} u, v \rhd_{\#} c_1, c_2 | h_1, h_2 \qquad u, v \rhd_{\#} l_1, l_2 | c_1 \barwedge h_1, c_2 \barwedge h_2 \\ u, v \rhd_{\#} r_1, r_2 | (\neg c_1) \barwedge h_1, (\neg c_2) \barwedge h_2 \end{array}}{u, v \rhd_{\#} \mathbf{If}\ c_1\ \mathbf{then}\ l_1\ \mathbf{else}\ r_1, \mathbf{If}\ c_2\ \mathbf{then}\ l_2\ \mathbf{else}\ r_2 | h_1, h_2}$$

$$\textsc{DeduceNonce}\ \frac{\mathsf{n} \blacktriangleright u | h_1 \qquad \mathsf{n} \blacktriangleright v | h_2}{u, v \rhd_{\#} \mathsf{n}, \mathsf{n} | h_1, h_2}$$

where $\mathsf{n} \blacktriangleright u|h$ is $\mathsf{n} \blacktriangleright u$ under the assumption that h holds.

Figure 5: Selected bi-deduction rules

To handle terms like $\mathsf{m}@_p\, \mathsf{pred}(T)$, the naive approach would unfold the macro. However, unfolding depends on the current trace, not just m and T, which is unacceptable since we require proofs to hold for any trace. Fortunately, [5, 45] already provide procedures for computing subterms in such cases, which we reuse. Thus, the goal $\textsc{VampPrf}_{\mathsf{m},p}(m, \mathsf{k}, T)$ simply offloads the search to [45].

In practice, we also extend the $m, \mathsf{k}, \mathsf{n} \blacktriangleright_{PRF} \_$ rules to include a guard condition, like to the bi-deduction rule introduced next.

5.2.4 An Improved Bi-Deduction Procedure. CryptoVampire2 can only close a goal using the following rules:

$$\textsc{Axiom}\ \frac{}{u \sim v \vdash u \sim v} \qquad \textsc{Refl}\ \frac{}{u \sim v \vdash a \sim a} \qquad \textsc{Deduce}\ \frac{u, v \rhd_{\#} a, b | \mathsf{true}, \mathsf{true}}{u \sim v \vdash a \sim b}$$

The rules Axiom and Refl are standard. Refl differs lightly from [5], allowing for macros in a as ours are tagged by their protocol.

However, the rule Deduce leaves us with a new proof obligation for *bi-deduction* [6]. Bi-deduction proofs are relevant to indistinguishability goals of the form $u \sim v \vdash a \sim b$. They express that a and b do not provide more information than u and v in the sense that we can compute a (resp. b) from u (resp. v) using the *same* polynomial-time algorithm $\mathcal{B}$. In [6], $\mathcal{B}$ is restricted to the attacker's randomness $\rho_{\mathcal{A}}$, meaning it cannot guess fresh nonces. We relax some of this restriction by extending bi-deduction to allow $\mathcal{B}$ limited but sufficient access to the honest tape ρ_h, culminating in the added rule DeduceNonce over the typical Squirrel rules.

Fig. 5 presents a relevant subset of the bi-deduction rules implemented in CryptoVampire2. These rules are powerful as they do not require additional context: proving bi-deducibility can be done without knowledge of the rest of the protocol, leading to proofs with few choices and much goal reuse.

Let us formally define our new bi-deduction. Let $\pi_H(\rho)$, with $H \subseteq \mathbb{N}$ and ρ a tape, be the projection of ρ induced by taking its value at each index in H. And we write $\mathsf{stable}^{\eta}_{\mathbb{M},\mathbb{T}}(h, H)$ when the term h is *stable* by $H \subseteq \mathbb{N}$, meaning $[\![h]\!]^{\eta}_{\mathbb{M},\mathbb{T}}((\rho_h, \rho_{\mathcal{A}}))$ keeps its value when changing arbitrarily many bits of ρ_h whose indices are in H. The notion is formalized in the technical report [46] . We can then formally define our new bi-deduction:

Definition 14 (Bi-Deduction). *Given $u_1^*, u_2^*, v_1^*, v_2^*, h_1^*, h_2^* \in \mathcal{S}$, we say that $u_1^*, u_2^* \triangleright_\# v_1^*, v_2^* | h_1^*, h_2^*$ hold when for all h_1, h_2 such that $h_i \equiv h_i^*$, we can find*

(1) u_1, u_2, v_1, v_2 terms
(2) a sequence $(H_\eta)_{\eta \in \mathbb{N}}$ of subsets of $\mathbb{N}$ of size N_η;
(3) $\mathcal{B}$, a polynomial-time Turing machine

such that

$$v_i \equiv v_i^* \text{ and } u_i \equiv \textbf{If } h_i \textbf{ then } u_i^* \textbf{ else } \emptyset \tag{33}$$

$$\text{stable}^\eta_{\mathbb{M},\mathbb{T}}(h_i, H_\eta) \Rightarrow \text{stable}^\eta_{\mathbb{M},\mathbb{T}}(u_i, H_\eta) \tag{34}$$

$$\mathcal{B}\left(1^\eta, \rho_{\mathcal{A}}, \pi_{H_\eta}(\rho_h), [\![u_i]\!]^\eta(\rho)\right) = [\![v_i]\!]^\eta(\rho) \text{ over } R_\eta^i \tag{35}$$

where $R_\eta^i = \{[\![h_i]\!]^\eta(\rho) = 1\}$ and N_η is a polynomial in η

In less formal terms: $\left(R_\eta^i\right)_{n \in \mathbb{N}}$ restricts the space to where the hypothesis h_i holds. π_{H_η} (Item 2) selects parts of the random tape irrelevant to the computation (Equation 34). H_η notably selects the nonces of v_1 and v_2 not used in u_1 and u_2. $\mathcal{B}$ and (35) allow computing v_1 (resp. v_2) from u_1 (resp. u_2) with access to the attacker tape and the selected part of the *honest* random tape when h_1 (resp. h_2) holds. DeduceNonce and (34) ensure that the selected part of ρ_h is independent from the u_i's computation, letting us share it with the distinguisher $\mathcal{B}$. This notion yields:

Property 7 (Bi-Deduction). *Deduce and DeduceNonce hold.*

5.2.5 Enriching the Frame. Deduce is sometimes insufficient to overcome the limitation of the weak induction hypothesis given by the Induction rule. This notably happens when DeduceNonce encounters a nonce n that is not fresh. This is typically overcome by attempting to induct over (36) instead.

$$\langle \text{frame}@p_1\tau, \mathsf{n} \rangle \sim \langle \text{frame}@p_2\tau, \mathsf{n} \rangle \tag{36}$$

Although CryptoVampire2 does not support goals of this form, we can emulate such a proof technique by proving a weaker protocol which "leaks" n in init or using Rmk. 1's extra init steps.

On failure, CryptoVampire2 can retry a proof with a frame enriched with some nonces. It will guide its choices by remembering which nonces blocked the previous proof because of DeduceNonce. This approach often finds a suitable set of nonces much faster than basic exhaustive search (Table 1).

5.3 Quantifiers and Variables

One major roadblock toward automating ours and Squirrel's peculiar fragment of the CCSA comes in the form of *binders*. As a reminder from Section 3.2's Fig. 2, they come in the form of the lookups **Find** $\vec{i}, \vec{\tau}$ **st**. t **then** t' **else** t'' and can be used to further define objects similar to the first-order quantifiers $\bar{\exists}$ and $\bar{\forall}$ as shown in (13) and (14) respectively. These are higher-order terms which neither of our backends (i.e., egg and Smt-Lib solvers) natively support. This section explores the transformations required to sufficiently capture their semantics throughout CryptoVampire2.

5.3.1 In First Order Logic. [45] already presents and uses results to soundly transform $\bar{\exists}$ and $\bar{\forall}$ into their first order logic counterparts. Such equivalence does not exist for general lookups. In [45], lookups are each replaced by free functions leveraging Skolemization [65] whose semantics are then described through added axioms.

Example 19 (Naming of Lookups). *Let $X \coloneqq$ **Find** $\vec{\beta}$ **st**. t **then** t' **else** t'' be a lookup with $\vec{\alpha} \coloneqq \text{fv}(X)$. We introduce the free function find_X and the skolems $\vec{\text{sk}}_X$ such that:*

$$\text{find}_X\left(\vec{\alpha}, \vec{\beta}\right) \coloneqq \textbf{If } t \textbf{ then } t' \textbf{ else } t'' \tag{37}$$

$$\forall \vec{\alpha}, \vec{\beta}.\ t \Rightarrow t\left\{\vec{\beta} \mapsto \vec{\text{sk}}_X(\vec{\alpha})\right\} \tag{38}$$

We then use $\text{find}_X\left(\vec{\alpha}, \vec{\text{sk}}_X(\vec{\alpha})\right)$ in lieu of X in first-order formulas.

However, unlike in [45], we do not know ahead of time which lookups are part of the problem. Moreover, as this is a fairly expensive transformation to the SMT solver, we want to reuse as many of the "named" functions as possible. To that end, CryptoVampire2 implements *caching* and *quantifier unification*.

5.3.2 In the E-Graph. One could use the same strategy within the e-graph. This, however, prevents rewriting *within* a binder. Instead, we take inspiration from [63] and enable our e-graph to reason about any binder using Modal De Bruijn indices [31] and Peano arithmetic [39]. We introduce the usual two symbols to represent variables 0_λ and S_λ as the zero and successor function respectively. Thus, for instance, $\exists x, \exists y, g(x, y)$ becomes $\exists_\lambda(\exists_\lambda(g(S_\lambda(0_\lambda), 0_\lambda)))$. Moreover, we make S_λ a Modal Operator, in the sense that it applies to arbitrary terms and not just De Bruijn variables, and it intuitively increments all the variables of its arguments. Effectively, we include the following equality rule:

$$S_\lambda(f(t_1, \ldots, t_n)) \rightsquigarrow f(S_\lambda(t_1), \ldots, S_\lambda(t_n)) \tag{39}$$

Such encoding has the benefit of making free variables explicit. This is important as Section 5.2.1's marker functions can "extract" a term out of its binder, so can rewritings (e.g., $x \equiv_{\mathfrak{G}} \pi_1(\langle x, 0_\lambda \rangle)$). In these cases, soundness requires us to treat those free variables as universally quantified (or give up). Encoding bound variables using De Bruijn indices gives us this for free.

5.4 Soundness

The semantics of CryptoVampire2 are directly grounded in traditional cryptography (Section 3). This has the interesting feature that each rule can be proven separately while still retaining Th. 4.

Theorem 4 (Soundness of CryptoVampire2). *Let p_1 and p_2 be two compatible protocols using steps from $\mathcal{S}$. If CryptoVampire2 returns* true*, then p_1 and p_2 are* indistinguishable *according to Def. 7 in the computational model under consideration.*

Proof. Since all inference and equality rules are proven sound, Th. 3 lets us prove Def. 7's (21). □

6 Benchmarks

6.1 Implementation

CryptoVampire2 is available at [47]. It is implemented in around 26 000 lines of rust [1] code (1 600 for golgge). It contains a modified version of egg [67] and a reimplementation of [45] to fit the new shape of formulas. User input is handled through the steel [64] interpreter for Scheme [2], to which CryptoVampire2-specific bindings were added. This lets the user easily programmatically define their protocol while keeping most of the advantages of a domain-specific language. CryptoVampire2 offloads some goals to SMT

Protocol		Features Enabled	Enriched Nonce Sets	Memoization Hit Rate	Runtime (s)			Longest SMT Run (ms)			
					GOLGGE	SMT	Total	All	VAMPIRE	Z3	CVC5
basic_hash [20] △		PRF	-	24.0 %	0.1	1.4	1.6	30.8	96.4	**31.0**	421.3
hash_lock [48] △		PRF	-	27.4 %	0.3	2.1	2.3	79.3	**137.9**	142.1	✗
lak_tag [43] △		PRF, FA	-	40.8 %	5.3	9.0	14.4	20.9	454.7	**21.1**	142.1
mw [59] △		PRF, XOR, FA	-	44.7 %	16.0	8.7	24.7	22.9	125.5	**27.7**	451.0
feldhofer [37] △		ENK-KP, FA	1	48.3 %	110.6	19.9	130.5	15.0	34.6	**18.4**	48.8
private_auth [10] △		ENK-KP, IND-CCA, FA	-	35.9 %	0.9	3.9	4.9	19.1	**11.7**	35.4	✗
ddh [44]	base △	DDH, FA	3	40.4 %	2.3	11.3	13.6	14.6	**32.2**	37.6	✗
	base ■		3	34.6 %	2.6	14.3	17.0	17.7	**34.1**	35.8	✗
	composition △†		1	23.1 %	1.9	11.8	13.6	28.7	**16.5**	85.8	✗
	composition ■†		3	25.5 %	15.8	94.1	109.9	20.9	**33.8**	65.7	✗
ssh [53]	part 1 △†	DDH, FA	2	31.5 %	2.4	11.5	13.9	22.0	**33.9**	38.6	402.8
	part 2 △†		2	31.1 %	5.3	24.6	29.9	28.1	**39.3**	39.7	✗
	part 1 ■†		-	32.8 %	0.5	4.9	5.4	11.8	29.4	**23.4**	55.2
	part 2 ■†		-	15.7 %	0.2	1.5	1.7	9.6	23.4	**14.0**	19.9
lfmtp ■‡		PRF	-	41.1 %	0.6	4.2	4.8	11.2	30.0	**19.6**	68.4
toy_state ■‡		PRF	-	27.5 %	0.1	0.4	0.5	10.3	32.9	**14.5**	66.8

‡ stateful
† using the composition framework [27]
✗ fails to prove indistinguishability reguardless of timeout
△ unlinkability query
■ strong secrecy query

Table 1: Summary of the Experiments

solvers (as in [45]) via SMT-LIB [13], using VAMPIRE [51], Z3 [33] and CVC5 [12] in parallel.

6.2 Setup and Limitations

6.2.1 Experimental Setup. We used a 24-thread machine equipped with an AMD 9900X processor and 64 GB of RAM, less than 1.4 GiB of which was required for all experiments. We used VAMPIRE 4.9, Z3 4.15.8 and CVC5 1.3.3. All solver timeouts were set to 150 ms (the lowest allowed by VAMPIRE) for the "Runtime" column of Table 1 and 750 ms[7] for the "Longest SMT Run" column.

6.2.2 Description of the Benchmarks. We evaluated CRYPTOVAMPIRE2 on all supported indistinguishability goals from SQUIRREL's benchmarks [5, 6]. This means all queries that could be converted and expressed as $\mathsf{frame}@_{p_1} t \sim \mathsf{frame}@_{p_2} t$. This includes stateful protocols and protocols using the composition framework [27], but it notably excludes yubikey-related lemmas. Overall, they consist of unlinkability and anonymity properties in the basic_hash [20], hash_lock [48], lak_tag [43], feldhofer [37], mw [59], private auth [10], ddh [44] and ssh [53] protocols; strong secrecy of the key in ddh, ssh; and two toyish stateful protocols from SQUIRREL (lfmtp and toy_state). We closely follow the modeling in [5, 6].

All experimental results are summarized in Table 1. It captures the sort of queries tested (unlinkability or strong secrecy) along with the features required of CRYPTOVAMPIRE2 to prove the query. That is, the notable cryptographic assumptions (see the technical report [46] for detailed rules) and proof techniques required (e.g., usage of Section 5.2.5, and to which extent). We further recorded the running times split between the SMT solver and GOLGGE (i.e., the rest of CRYPTOVAMPIRE2), and the longest time taken by each SMT backend to prove a query.

[7] chosen such that the result of the overall proof is not influenced by the timeout

6.2.3 Limitations. To automate indistinguishability proofs, CRYPTOVAMPIRE2 sometimes requires *trace* lemmas as input. In particular, all stateless protocols except private_auth require one trace lemma capturing the authentication properties expected by their specification (e.g., Lem. 1 for basic_hash), while both stateful protocols require secrecy lemmas about their state. These lemmas constrain the search space and preserve execution alignment during the proof. They are not arbitrary proof artifacts, but security goals of the protocol specification. While synthesizing such invariants is future work, tools like [45] can already verify the authentication lemmas when provided, and SQUIRREL can verify all of them.

As mentioned above, CRYPTOVAMPIRE2 is currently strictly constrained to bi-processes. Effectively, this restricts our support for stateful protocols, excluding 11 indistinguishability goals from [6]. These goals are typically lemmas for broader trace properties.

Furthermore, here we consider only examples from [5, 6]. We do not take into account later extensions made to SQUIRREL.

6.3 Analysis

6.3.1 General Runtime Analysis. Despite the short timeout, we observe that almost all the runtime is spent waiting for the SMT solvers. Deeper analysis reveals that most of it is spent on impossible queries waiting for the solvers to time out. This underscores the importance of the longest SMT query in CRYPTOVAMPIRE2's runtime, as this dictates the timeout that can be given to the SMT backend. Moreover, we observe that the proportion of time spent on SMT proving reduces as the total runtime increases. This correlates with increased memoization: as time goes on, SMT queries, or goals that would lead to some, are cached.

6.3.2 Backend Comparison. We compare the SMT backends (each solver individually, then all together) on their ability to terminate

quickly on all successful queries. This metric is crucial, as it determines the minimum timeout required for a configuration to prove indistinguishability, which in turn has a significant impact on the total runtime. We manually verified that failing cvc5 runs were not due to timeouts, but rather to incomplete solver strategies. The fastest single solver is marked in **bold** in Table 1.

Interestingly, Vampire alone sometimes outperforms racing all solvers in parallel (including Vampire itself) for a result. This paradoxical outcome is explained by a different resource allocation when all solvers are enabled. Such cases are marked with underlines.

One should note, however, that CryptoVampire2 runs the solvers in close-to-default configurations. Their performance could likely be significantly improved with further tuning.

Overall, Vampire is more often the fastest, while z3 is not far behind with more consistent runtimes. The poor performance of cvc5 contrasts with the results of [45]. This highlights the high volatility of SMT solvers, which justifies why CryptoVampire2 does not rely on a single solver.

7 Conclusion and Future Work

This paper has introduced CryptoVampire2, the first auto-active verifier[8] for indistinguishability in the computational model using the CCSA framework. To that end, we also introduced golgge—a novel backtracking engine supported by e-graphs—which enables a more relaxed approach to equality, key to CryptoVampire2's success. CryptoVampire2 was furthermore tested against the examples of the state-of-the-art proof assistant Squirrel [5], successfully proving them all.

Future work will focus on extending CryptoVampire2 to advanced CCSA techniques such as quantum soundness [28]. Beyond such extensions, we aim to propagate the new equality reasoning philosophy to [45], alongside various improved heuristics and optimizations for CryptoVampire2 and golgge.

Acknowledgments

This work was partially supported by the European Research Council (ERC) through the Consolidator Grants ARTIST 101002685 and BlockSec 101141432; 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]; and the Austrian Research Promotion Agency (FFG) through the COMET K1 SBA.

This paper was edited for grammar and phrasing using Mistral, Gemini, ChatGPT, and various versions of Qwen. All outputs were inspected by the authors to ensure accuracy and originality.

[8] i.e., a largely automated verifier requiring some user input, in our case, some lemmas.

References

[1] [n. d.]. *Rust Programming Language*. https://rust-lang.org/
[2] [n. d.]. *The Scheme Programming Language*. https://www.scheme.org/
[3] Nadhem J. Al Fardan and Kenneth G. Paterson. 2013. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy* (2013-05). 526–540. doi:10.1109/SP.2013.42
[4] Thibaut Antoine and David Baelde. 2025. Propositional Logics of Overwhelming Truth. In *CSL 2025 - 33rd EACSL Annual Conference on Computer Science Logic* (Amsterdam, Netherlands, 2025-02). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2025.24
[5] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. 2021. An Interactive Prover for Protocol Verification in the Computational Model. In *2021 IEEE Symposium on Security and Privacy (SP)* (San Fransisco / Virtual, United States, 2021-05). 537–554. doi:10.1109/SP40001.2021.00078
[6] David Baelde, Stéphanie Delaune, Adrien Koutsos, and Solène Moreau. 2022. Cracking the Stateful Nut: Computational Proofs of Stateful Security Protocols Using the Squirrel Proof Assistant. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)* (Haifa, Israel, 2022-08). 289–304. doi:10.1109/CSF54842.2022.9919665
[7] David Baelde, Stéphanie Delaune, and Stanislas Riou. 2025. SMT-based Automation for Overwhelming Truth. In *38th IEEE Computer Security Foundations Symposium* (Santa Cruz, United States, 2025-06). IEEE, 505–520. doi:10.1109/CSF64896.2025.00019
[8] David Baelde, Adrien Koutsos, and Justine Sauvage. 2024. Foundations for Cryptographic Reductions in CCSA Logics. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2024-12-09) *(CCS '24)*. Association for Computing Machinery, 2814–2828. doi:10.1145/3658644.3690193
[9] Gergei Bana and Hubert Comon-Lundh. 2012. Towards Unconditional Soundness: Computationally Complete Symbolic Attacker. In *Principles of Security and Trust* (Berlin, Heidelberg, 2012) *(Lecture Notes in Computer Science)*, Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer, 189–208. doi:10.1007/978-3-642-28641-4_11
[10] Gergei Bana and Hubert Comon-Lundh. 2014. A Computationally Complete Symbolic Attacker for Equivalence Properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014). 609–620.
[11] Gergei Bana, Koji Hasebe, and Mitsuhiro Okada. 2013. Computationally Complete Symbolic Attacker and Key Exchange. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer \& Communications Security* (2013) *(CCS '13)*. Association for Computing Machinery, 1231–1246. doi:10.1145/2508859.2516710
[12] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. 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* (2022) *(Lecture Notes in Computer Science, Vol. 13243)*, Dana Fisman and Grigore Rosu (Eds.). Springer, 415–442. doi:10.1007/978-3-030-99524-9_24
[13] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2017. The SMT-LIB Standard: Version 2.6.
[14] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. 2011. Computer-Aided Security Proofs for the Working Cryptographer. In *Advances in Cryptology – CRYPTO 2011* (Berlin, Heidelberg, 2011) *(Lecture Notes in Computer Science)*, Phillip Rogaway (Ed.). Springer, 71–90. doi:10.1007/978-3-642-22792-9_5
[15] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. 2014. Proving the TLS Handshake Secure (As It Is). In *Advances in Cryptology – CRYPTO 2014* (2014). Springer, Berlin, Heidelberg, 235–255. doi:10.1007/978-3-662-44381-1_14
[16] Karthikeyan Bhargavan and Gaetan Leurent. 2016. Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH. In *Proceedings 2016 Network and Distributed System Security Symposium* (San Diego, CA, 2016). Internet Society. doi:10.14722/ndss.2016.23418
[17] Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)* (Cape Breton, Nova Scotia, Canada, 2001-06). IEEE Computer Society, 82–96. doi:10.1109/CSFW.2001.930138
[18] B. Blanchet. 2006. A Computationally Sound Mechanized Prover for Security Protocols. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (2006-05). 15 pp.–154. doi:10.1109/SP.2006.1
[19] Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2008. Automated Verification of Selected Equivalences for Security Protocols. 75, 1 (2008), 3–51. doi:10.1016/j.jlap.2007.06.002
[20] Mayla Brusó, Konstantinos Chatzikokolakis, and Jerry den Hartog. 2010. Formal Verification of Privacy for RFID Systems. In *2010 23rd IEEE Computer Security*

Foundations Symposium (2010-07). 75–88. doi:10.1109/CSF.2010.13
[21] Alan Bundy. 2001. Chapter 13 - The Automation of Proof by Mathematical Induction. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). North-Holland, 845–911. doi:10.1016/B978-044450813-3/50015-1
[22] Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1989. What You Always Wanted to Know about Datalog (and Never Dared to Ask). 1, 1 (1989), 146–166. doi:10.1109/69.43410
[23] Weidong Chen and David S. Warren. 1996. Tabled Evaluation with Delaying for General Logic Programs. 43, 1 (1996), 20–74. doi:10.1145/227595.227597
[24] Vincent Cheval and Bruno Blanchet. 2013. Proving More Observational Equivalences with ProVerif. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 226–246. doi:10.1007/978-3-642-36830-1_12
[25] Michael R. Clarkson and Fred B. Schneider. 2008. Hyperproperties. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, USA, 23-25 June 2008* (2008). IEEE Computer Society, 51–65. doi:10.1109/CSF.2008.7
[26] Alain Colmerauer, Henri Kanoui, Robert Pasero, and Philippe Roussel. 1973. Un systeme de communication homme-machine en francais. (1973).
[27] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. 2020. 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, 2020-10-30). ACM, 1427–1444. doi:10.1145/3372297.3417229
[28] Cas Cremers, Caroline Fontaine, and Charlie Jacomme. 2022. A Logic and an Interactive Prover for the Computational Post-Quantum Security of Protocols. In *2022 IEEE Symposium on Security and Privacy (SP)* (2022-05). 125–141. doi:10.1109/SP46214.2022.9833800
[29] Baelde David, Adrien Koutsos, and Joseph Lallemand. 2023. A Higher-Order Indistinguishability Logic for Cryptographic Reasoning. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (2023-06). 1–13. doi:10.1109/LICS56636.2023.10175781
[30] Martin Davis, George Logemann, and Donald Loveland. 1962. A Machine Program for Theorem-Proving. 5, 7 (1962), 394–397. doi:10.1145/368273.368557
[31] Nicolaas Govert de Bruijn. 1972. Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. 75, 5 (1972), 381–392. doi:10.1016/1385-7258(72)90034-0
[32] Leonardo de Moura and Nikolaj Bjø rner. 2007. Efficient E-Matching for SMT Solvers. In *Automated Deduction – CADE-21* (Berlin, Heidelberg, 2007), Frank Pfenning (Ed.). Springer, 183–198. doi:10.1007/978-3-540-73595-3_13
[33] Leonardo de Moura and Nikolaj Bjø rner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems* (Berlin, Heidelberg, 2008) *(Lecture Notes in Computer Science)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. doi:10.1007/978-3-540-78800-3_24
[34] Stéphanie Delaune, Clément Herouard, and Joseph Lallemand. 2025. Secrecy by Typing in the Computational Model. 17–32. doi:10.1109/CSF64896.2025.00006
[35] D. Dolev and A. Yao. 1983. On the Security of Public Key Protocols. 29, 2 (1983), 198–208. doi:10.1109/TIT.1983.1056650
[36] Clemens Eisenhofer, Ruba Alassaf, Michael Rawson, and Laura Kovács. 2023. Non-Classical Logics in Satisfiability Modulo Theories. In *Automated Reasoning with Analytic Tableaux and Related Methods* (Cham, 2023) *(Lecture Notes in Computer Science)*, Revantha Ramanayake and Josef Urban (Eds.). Springer Nature Switzerland, 24–36. doi:10.1007/978-3-031-43513-3_2
[37] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. 2004. Strong Authentication for RFID Systems Using the AES Algorithm. In *Cryptographic Hardware and Embedded Systems - CHES 2004* (Berlin, Heidelberg, 2004) *(Lecture Notes in Computer Science)*, Marc Joye and Jean-Jacques Quisquater (Eds.). Springer, 357–370. doi:10.1007/978-3-540-28632-5_26
[38] Joshua Gancher, Sydney Gibson, Pratap Singh, Samvid Dharanikota, and Bryan Parno. 2023. Owl: Compositional Verification of Security Protocols via an Information-Flow Type System. In *2023 IEEE Symposium on Security and Privacy (SP)* (2023-05). 1130–1147. doi:10.1109/SP46215.2023.10179477
[39] Giuseppe Peano. 1889. *Arithmetices principia: nova methodo*. Fratres Bocca.
[40] Oded Goldreich. 2007. Foundations of Cryptography. Vol. 1: Basic Tools. Cambridge Univ. Press.
[41] Márton Hajdú, Petra Hozzová, Laura Kovács, Johannes Schoisswohl, and Andrei Voronkov. 2020. Induction with Generalization in Superposition Reasoning. In *Intelligent Computer Mathematics* (Cham, 2020) *(Lecture Notes in Computer Science)*, Christoph Benzmüller and Bruce Miller (Eds.). Springer International Publishing, 123–137. doi:10.1007/978-3-030-53518-6_8
[42] Jacques Herbrand. 1930. *Recherches sur la théorie de la démonstration*.
[43] Lucca Hirschi, David Baelde, and Stéphanie Delaune. 2016. A Method for Verifying Privacy-Type Properties: The Unbounded Case. In *2016 IEEE Symposium on Security and Privacy (SP)* (2016-05). 564–581. doi:10.1109/SP.2016.40
[44] ISO/IEC 9798-3:2019 2019. IT Security Techniques – Entity Authentication – Part 3: Mechanisms Using Digital Signature Techniques.
[45] Simon Jeanteur, Laura Kovács, Matteo Maffei, and Michael Rawson. 2024. CryptoVampire: Automated Reasoning for the Complete Symbolic Attacker Cryptographic Model. IEEE Computer Society, 3165–3183. doi:10.1109/SP54263.2024.00246
[46] Simon Jeanteur, Laura Kovács, Matteo Maffei, and Michael Rawson. 2026. *Automated Reasoning for Indistinguishability in the CCSA*. https://eprint.iacr.org/2026/1853
[47] Simon Jeanteur, Laura Kovàcs, Matteo Maffei, and Michael Rawson. 2026. *Cryptovampire*. https://github.com/SecPriv/CryptoVampire
[48] Ari Juels and Stephen A. Weis. 2009. Defining Strong Privacy for RFID. 13, 1 (2009), 7:1–7:23. doi:10.1145/1609956.1609963
[49] Donald E. Knuth and Petter B. Bendix. 1970. Simple Word Problems in Universal Algebras. In *Computational Problems in Abstract Algebra*, John Leech (Ed.). Pergamon, 263–297. doi:10.1016/B978-0-08-012975-4.50028-X
[50] Adrien Koutsos. 2019. The 5G-AKA Authentication Protocol Privacy. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* (2019-06). 464–479. doi:10.1109/EuroSP.2019.00041
[51] Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In *Computer Aided Verification* (Berlin, Heidelberg, 2013) *(Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 1–35. doi:10.1007/978-3-642-39799-8_1
[52] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. 2019. A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* (2019-06). 231–246. doi:10.1109/EuroSP.2019.00026
[53] Chris M. Lonvick and Tatu Ylonen. 2006. The Secure Shell (SSH) Transport Layer Protocol. doi:10.17487/RFC4253
[54] Gavin Lowe. 1995. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. 56, 3 (1995), 131–133. doi:10.1016/0020-0190(95)00144-2
[55] G. Lowe. 1997. A Hierarchy of Authentication Specifications. In *Proceedings 10th Computer Security Foundations Workshop* (1997-06). 31–43. doi:10.1109/CSFW.1997.596782
[56] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *Proceedings 2019 Network and Distributed System Security Symposium* (San Diego, CA, 2019). Internet Society. doi:10.14722/ndss.2019.23330
[57] J.P. Marques Silva and K.A. Sakallah. 1996. GRASP-A New Search Algorithm for Satisfiability. In *Proceedings of International Conference on Computer Aided Design* (1996-11). 220–227. doi:10.1109/ICCAD.1996.569607
[58] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification* (Berlin, Heidelberg, 2013) *(Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 696–701. doi:10.1007/978-3-642-39799-8_48
[59] David Molnar and David Wagner. 2004. Privacy and Security in Library RFID: Issues, Practices, and Architectures. In *Proceedings of the 11th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2004-10-25) *(CCS '04)*. Association for Computing Machinery, 210–219. doi:10.1145/1030083.1030112
[60] Charles Gregory Nelson. 1980. *Techniques for Program Verification*. Stanford University.
[61] Greg Nelson and Derek C. Oppen. 1980. Fast Decision Procedures Based on Congruence Closure. 27, 2 (1980), 356–364. doi:10.1145/322186.322198
[62] Jens Otten. 2008. leanCoP 2.0and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic (System Descriptions). In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings* (2008) *(Lecture Notes in Computer Science, Vol. 5195)*, Alessandro Armando, Peter Baumgartner, and Gilles Dowek (Eds.). Springer, 283–291. doi:10.1007/978-3-540-71070-7_23
[63] Pavel Panchekha. 2021. *Binding in E-graphs*. https://pavpanchekha.com/blog/egg-bindings.html
[64] Matthew Paras. 2025. *Mattwparas/Steel*. https://github.com/mattwparas/steel
[65] Thoralf Skolem. 1920. Logisch-Kombinatorische Untersuchungen Über Die Erfüllbarkeit Oder Bewiesbarkeit Mathematischer Sätze Nebst Einem Theorem Über Dichte Mengen. In *Selected Works in Logic*.
[66] Alfred Tarski. 1933. Pojęcie Prawdy w Językach Nauk Dedukcyjnych. (1933).
[67] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. Egg: Fast and Extensible Equality Saturation. 5, Article 23 (2021). Issue POPL. doi:10.1145/3434304
[68] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. 7 (2023), 125:468–125:492. Issue PLDI. doi:10.1145/3591239

Open Science

All code, benchmarks and instructions on how to use them are available at [47].

Ethical Considerations

The research presented in this paper focuses on the development of formal verification techniques to improve the security and privacy of cryptographic protocols. As a tool for automated theorem proving, its primary purpose is to provide rigorous guarantees of security, thereby contributing to the robustness of digital infrastructure. This research does not involve human subjects, personal data, or real-world vulnerability analysis of deployed systems. Our experimental evaluation relies on established formal models of protocols and standard cryptographic assumptions.

Unlike many protocol verifiers, CryptoVampire2 cannot find attacks. It could nonetheless still be used by nefarious actors to build highly resilient systems for malicious purposes. We believe, however, that the benefits of lowering the barrier to building secure protocol standards and implementations for the general public far outweigh these potential harms.

Generative AI Usage

Generative AI was used for editorial purposes on this paper, to orchestrate the testing (the tests themselves were written by hand). They were used to very little success within CryptoVampire2, hence the code itself was overwhelmingly written by hand.

In all cases, their output was *never* blindly trusted, and we always thoroughly checked for correctness.