Stop Calling Them Provers

The Case for Calling ProVerif and Tamarin Symbolic Protocol Analyzers

Nadim Kobeissi

Symbolic Software

Abstract. A claim that a cryptographic protocol has been proved secure depends on the theory, the analysis software, the protocol model, and the security property being checked. I examine each of these for ProVerif and Tamarin. An audit of their published soundness arguments finds false statements, missing proofs, and gaps between the procedures justified on paper and those the tools run. I also reproduce previously reported bugs that cause Tamarin to accept false claims about the submitted models. ProVerif supplies no certificate for its full analysis, while Tamarin replays its proof tree with its own solver, leaving the correctness of that code to be trusted.

A correct symbolic result can still fall short of the security claim made for a protocol. Models can omit behavior in parsers and state machines; secrecy queries can allow information to leak; and equations for cryptographic primitives need justification before they support computational guarantees. Better models, stronger queries and computational soundness arguments can address these problems, but someone must establish that they apply to the protocol at hand.

For these reasons and more, I argue that calling ProVerif and Tamarin "provers" is misleading: we should call them symbolic protocol analyzers and report their results with the models, assumptions and evidence that support them. They remain useful for finding attacks and establishing symbolic properties, including for unbounded sessions.

Keywords: symbolic protocol analysis · formal verification · ProVerif · Tamarin · soundness · Dolev–Yao model · computational security

1 Introduction

ProVerif's manual tells a user how to read a successful result: "RESULT [Query] is true: The query is proved, there is no attack." [22, p. 31] Tamarin prints "verified", makes itself available through the domain name `tamarin-prover.com`, and its tool paper says this means that all protocol traces satisfy the formula [49]. The manuals are careful to explain that these are claims about the models the tools analyze. But a person using either tool to study a cryptographic protocol still has to decide what the answer means for that protocol. The word "proof" gives them less help than it appears to.

Consider a bug Sven Hammann reported in September 2026. Two processes read inputs into variables with the same name. One then decrypts an incoming message. Tamarin 1.12.0 translates the model so that decryption can never succeed, and verifies the claim that no message is ever decrypted [38]. Rename the other process's input variable, and the verdict changes. Other reported bugs lose an attacker's access to a public channel, accept an impossible execution, or report two distinguishable processes as equivalent.

E-mail: `nadim@symbolic.software` (Nadim Kobeissi)

These are mistakes about the submitted model itself. They arise before anyone has to argue about whether the model describes the real world.

I argue that ProVerif and Tamarin should be called *symbolic protocol analyzers*. They can find attacks and establish useful facts about models, sometimes for arbitrarily many sessions. But a reader should also be able to check how a positive answer follows from the submitted model. Neither tool produces an independently checked proof covering every step from its input to its answer. The audit below asks how far the published arguments justify those steps and where the running programs get them wrong. Calling the tool a prover does not settle either question.

Symbolic statements can have proofs, including proofs checked by a small kernel. The title argues against calling these tools cryptographic protocol provers on the strength of the results they return. I would apply the same standard to a computational tool: show the argument, say what it proves, and account for the connection to the system being discussed. Koblitz and Menezes have long challenged how computational "provable security" is presented [44], and Degabriele, Paterson and Watson describe attacks outside the models of computational proofs [32]. Our choice of formalism cannot settle which assumptions are justified.

A bad model can be given to any prover. That fact does no work in my case for renaming these tools. It does matter when reporting what a particular result establishes: a theorem about a model may not support the security claim made for the protocol. I discuss that separate question to explain what a report should say, including how better models and queries can repair such mistakes.

The distinctions themselves are familiar. Barbosa et al.'s survey separates security claims about designs, implementations and the tools trusted to verify them [8]. What follows puts those distinctions to work on particular claims. Sections 2 and 3 explain what the tools produce and what a reader can check. Section 4 audits the published arguments and reproduces selected bugs, with credit to their reporters. Section 5 assumes a correct analyzer and examines what its result says about the protocol. Section 6 proposes how to report the results.

2 Saturation and constraint solving

To assess a positive result, we need to know how the tool reached it. Both ProVerif and Tamarin can establish a claim about all executions by ruling out counterexamples. If the rules cover every possible counterexample and the procedure excludes them correctly, that is a valid proof. The difficulty is connecting that description to the particular program run whose output we are reading.

ProVerif first translates the protocol and the attacker into Horn clauses, which are rules saying what follows from what. The fact $\mathsf{attacker}(M)$ means that the attacker may know message M. Fresh values become functions of other messages, and the translation forgets how many times messages have been used. These abstractions allow the rules to describe arbitrarily many protocol sessions [17]. ProVerif combines the clauses by resolution until it has covered the required consequences, a process called saturation. If saturation finishes and the attacker cannot derive the secret s, ProVerif reports that s is secret.

The founding paper states the intended guarantee: "Our approximations are safe, in the sense that if the verifier does not find a flaw in the protocol, then there is no flaw. The verifier therefore provides real security guarantees." [17, p. 1] At the time, the same paper said that the proof connecting processes to clauses was still in progress [17, p. 5]. Later papers supplied translations and soundness results for secrecy, correspondences, equivalence, lemmas and induction [1, 21, 16, 19]. A soundness result justifies drawing the conclusion when its assumptions cover the process and procedure being used.

If the clauses do derive an attack, ProVerif tries to reconstruct an execution of the process. When it succeeds, it reports “is false”. When it cannot, it reports “cannot be proved”, because the abstraction may have introduced an execution that the process cannot perform. The procedure can also fail to terminate. So “is true”, “cannot be proved” and a timeout mean different things, even if none comes with an attack trace.

Tamarin starts from multiset rewriting rules: rules that consume and produce facts describing the protocol’s state. A trace records the events that occur as these rules run, and a security property is a formula about traces. To check a universal property, Tamarin searches for a trace that violates it. Its constraint systems describe possible executions, and the search branches as it considers how their constraints could be satisfied. A solved system describes an attack; a finite tree with every branch closed rules out all the represented attacks. Meier’s thesis calls the latter “the finite and complete search tree without a Solved-node” [47, p. 149].

Users can guide this search through Tamarin’s interface, or write intermediate lemmas for it to prove and reuse [47, 58]. For a universal trace property, a successful exclusion gives “verified” and a counterexample gives “falsified”. Users can also ask whether an execution exists. That helps catch models in which the protocol cannot run, although it asks a different question from whether every execution is safe.

2.1 Finite proofs can cover unbounded sessions

A finite argument can establish a fact about an infinite collection of objects. We accept that in ordinary mathematics, and symbolic protocol analysis uses the same idea. ProVerif’s clauses describe arbitrarily many instances of protocol roles. Tamarin’s constraints describe families of executions, and its search can use induction. A successful finite analysis can therefore cover every finite number of modeled sessions without imposing a session cutoff [22, 58]. That is a substantial benefit, and nothing in this paper disputes it.

Nor does a procedure have to terminate on every input for its successful runs to prove something. The word “complete” in the account above concerns coverage of possible counterexamples. It does not promise a decision procedure that always finishes. ProVerif may also finish with “cannot be proved” for a true property, because its abstraction has lost too much precision. To trust a positive result, we need evidence that this run followed the justified rules and excluded every counterexample those rules represent.

The two tools supply different evidence for that last step. ProVerif’s positive verdict comes without an independently checkable certificate for the computation. Tamarin retains a tree that can be inspected and replayed, but replay uses its own solver. The next section examines what those choices ask us to trust. Even if that trust is justified, the result still concerns the executions and observations present in the model.

3 Proof certificates and independent checking

A proof is useful partly because someone other than its author can check it. A program’s announcement that a theorem holds gives the reader much less to work with. The reader must either trust that program or find another way to establish the result. In protocol analysis, the difference matters twice: we want to check the reasoning, and we want to understand what the theorem says about security.

Computer-assisted mathematics has dealt with both problems. Appel and Haken’s four-color proof relied on a large computation [3]. Gonthier writes that their proof and its later revision “combined a textual argument, which could reasonably be checked by inspection, with computer code that could not.” [35, p. 1] His Coq formalization made it possible for a different program to check the result. The formal proof of the Kepler

conjecture likewise reduced the risk of an error in a large computation to the risk of an error in a much smaller checking kernel [37]. Neither approach eliminates trust, but each makes the trusted part smaller and easier to examine.

I think a report claiming a proof of protocol security should let its reader do four things. First, inspect an argument for the result. Second, check that argument without relying entirely on the program that produced it; a small independent checker is the stronger version of this requirement, known as de Bruijn's criterion [9]. Third, read the definitions and assumptions that fix what the theorem says. Fourth, tell how far the reported security claim follows from that theorem. These are standards for the evidence a report supplies. They don't imply that a proof ceases to exist when a program fails to print it.

The last two requirements remain even when every inference has been checked. Gonthier observes that the skeptic "still needs to unravel all the definitions to convince himself that the result lives up to its name." [35, p. 7] A protocol theorem might call a value secret because the attacker cannot recover the whole value, while permitting the attacker to distinguish two candidates. Another theorem might exclude precisely the executions in which a nonce is reused. A checker can accept either theorem without telling the user whether it answers their question.

3.1 Positive verdicts lack independent checks

For a positive result, ProVerif's manual says: "The query is proved, there is no attack. In this case, ProVerif displays no attack derivation and no attack trace." [22, p. 31] The user receives the conclusion without a proof object. Goubault-Larrecq identified this problem in 2008, writing that "security is demonstrated through the absence of a proof" [36, p. 7]. He also developed a different procedure that could produce Coq proofs of secrecy at the clause level [36].

A certificate for saturation is possible in principle. A finite set of Horn clauses could let a checker verify that the starting clauses are covered, that the set is closed under the required inferences, and that the query is excluded. Redundant clauses can be handled through subsumption; selection, simplification and constraints need their own checks. These checks may be expensive, but they give a concrete account of what needs to be verified. There is then a further step: showing that the starting clauses correctly represent the user's process. ProVerif's ordinary positive output provides no independently checkable certificate covering this whole sequence.

Tamarin does provide an object. Its proof skeleton records the search tree and the methods used at each step [58, p. 147]. For a small model, a person can inspect that argument. This is more evidence than ProVerif's bare verdict. But when Tamarin replays the tree, it calls its own constraint solver, so a mistake in that solver may affect both the original search and its replay. There is no small independent checker separating the two.

Meier's thesis states the concern directly: "The correctness of results obtained using these tools obviously depends [sic] the soundness of the proofs justifying their underlying theory and the correctness of their implementations. Because of the complexity of these proofs and implementations, one may rightfully question the correctness of such results." [47, p. 15] His response for a smaller class of protocols was scyther-proof, which produces proofs checked in Isabelle [48]. DyLean now offers symbolic proofs checked in Lean [60]. These tools show that symbolic reasoning and independent checking are compatible.

3.2 Computational tools also need independent checks

A computational proof tool also needs a correct checker. CryptoVerif, EasyCrypt and Squirrel depend on their own software and, where used, external solvers; Barbosa et al. discuss these different trusted bases [8]. A general theorem prover's bare "unsatisfiable"

answer raises the same issue. The conventional name "prover" describes the program's intended function. It cannot tell a user whether the particular result in front of them is correct.

Comparing tools helps. Morio et al. found ProVerif bugs by translating models into another tool and comparing the results [55]. A disagreement gives researchers somewhere to investigate. Agreement provides evidence too, provided the inputs really describe the same problem, although shared assumptions can still make both answers wrong. Independent checking and comparison deserve credit for what they establish. Neither changes the meaning of the theorem being checked, and neither removes the need to examine how the published soundness argument reaches the running program. That is the subject of the audit below.

4 False verdicts and gaps in the soundness arguments

A soundness theorem tells us that a procedure's conclusions follow under its assumptions. To use it to justify a tool's answer, we have to check that the theorem covers the language, translations and search rules the tool actually used. The papers behind ProVerif and Tamarin contain substantial proofs, but following this connection through them reveals missing arguments and false statements. The implementation bugs show why the distinction matters: a core theorem can remain true while the program returns a wrong answer.

4.1 Paper audit and bug reproduction

I read the founding papers, later soundness papers, theses and extension papers cited below. For each, I checked which language and procedure its theorems cover, followed the arguments, and tested suspected false statements against the published definitions. The counterexamples in Section 4.5 were checked by hand, and executable cases were run with ProVerif 2.05 and Tamarin 1.12.0. Findings I could not confirm were excluded.

The implementation bugs reproduced below were reported by Hammann and Milner. I reproduce selected examples, add controls that isolate some of the failures, and connect the SAPIC+ and subterm examples to printed lemmas. The restriction-moving remark and Schmidt's rules are examined as written. For ProVerif's implementation history, I also use the release notes and Morio et al.'s cross-tool study [24, 55]. A historical fix is evidence that a bug existed; it is not evidence that version 2.05 still has it. This audit does not estimate either tool's defect rate or the number of published analyses affected.

Table 1 separates findings that should not be counted as if they were the same. An omitted condition that the code handles correctly matters less than a false "verified". I start with the latter, since these failures concern the submitted models themselves and need no argument about how well they represent a deployed protocol.

4.2 Bugs produce false security verdicts

In Hammann's decryption example, one process publishes a public key and accepts a ciphertext under it. The attacker can encrypt a message with that key and make the process decrypt it. Tamarin 1.12.0 nevertheless verifies that no message is ever decrypted, because the SAPIC translation has produced a decryption rule that cannot fire [38]. Another process uses the same input-variable name; renaming that variable restores the missing execution and flips the verdict. This is a translation error, and a proof about the correctly translated rules cannot justify the answer produced from the incorrect ones.

Another example publishes a channel through an alias, which the same version of Tamarin mishandles [39]:

Table 1: Findings grouped by what goes wrong. Recorded runs use ProVerif 2.05 and Tamarin 1.12.0.

Finding	Cases	Evidence
False security claim accepted	SAPIC variable clash and channel alias; natural subterm; diff restrictions	Prior reports, reproduced here.
Nonexistent execution accepted	Hidden subterm of an attacker input	Prior report, reproduced; printed lemma examined here.
Spurious attack reported	SAPIC pure state; hidden subterm	Prior reports, reproduced.
False printed statement	SAPIC+ and subterm lemmas; Schmidt rules; restriction-moving remark	Counterexamples to the statements as written. Tool behavior assessed separately.
Missing argument or narrower result	Translation links, equivalence lemmas, optimized relation	The papers leave a step unjustified; the result may still hold.
Condition omitted in the text	Induction base case; bound-variable convention	Code or later exposition supplies the condition.

```
new c; let x = c in out(x);
( out(c, 'ok') | in(c, y); event R(y) )
```

The process creates a channel c, gives it the alias x, and publishes x. The attacker therefore knows c and can send `'adv'` on it. Tamarin verifies instead that every received y is `'ok'`. Its secret-channel analysis runs before alias substitution and removes the rule that lets the attacker send on the channel. Publishing c directly gives the correct opposite verdict. Both versions still have an honest execution, so a check that the protocol can complete would not catch this loss of attacker behavior.

Milner's example uses the following multiset rewriting rule [52]:

```
[ In(%n) ] --[ A(f('a'), %n) ]-> []
```

The rule accepts a number n and records $A(f(\texttt{'a'}), n)$. The queried claim says that every event $A(x, n)$ has x as a proper subterm of n, meaning that x occurs inside n. Input 1 is a counterexample: $f(\texttt{'a'})$ does not occur inside 1. Tamarin 1.12.0 verifies the claim anyway. It also verifies an added query that the execution with input 1 exists. The reported cause is an arithmetic transformation applied to a term of the wrong sort; the failure depends on the natural-number and subterm features.

Existence checks are useful, but they still rely on the tool. The subterm bug in Section 4.5 goes further and accepts an execution that does not exist. So a successful security query and a successful executability query need not independently confirm each other. To gain independent evidence from an alleged execution, someone has to replay it against the input model.

Equivalence mode supplies another false positive:

```
rule Emit: [] --[ A(diff('a','b')) ]-> [ Out('hello') ]
restriction BanRight: "All #i. A('b')@i ==> F"
```

The left process records $A(\texttt{'a'})$ and sends `'hello'`. The right records $A(\texttt{'b'})$, which the restriction forbids, so it cannot send the message. The processes are distinguishable. Tamarin 1.12.0 reports "verified (36 steps)" for equivalence even though its side checks correctly find that only the left rule can run. The repair discussion explains that the tool compares the restriction with the left graph, finds no match, and drops it [54]. The cited equivalence theorem does not treat trace restrictions [10]. The bug falls outside that theorem, but it remains inside the feature set offered to the user.

ProVerif's history also contains soundness repairs. Version 2.02pl1 fixed uses of induction across phases and an interaction between restrictions and a public-channel optimization; version 2.04 corrected subsumption involving ground public terms [24]. These are fixes documented by the developers, not failures reproduced here on version 2.05. Morio,

Table 2: The steps from a ProVerif model to a positive verdict, and the arguments supplied for them.

Step	Source	What is supplied
Equations to constructor rewrite rules	2008 [16]	Proof for equivalence (Lemma 1). For secrecy and correspondence, a similar result is asserted but not detailed (p. 17).
Process to instrumented process	2022 [19]	Lemma 3, without proof.
Instrumented process to clauses	2022, Thm. 1	Proof for a core language whose constructors have only identity and failure rules (p. 36).
Saturation	2022, Thm. 2	Proof.
Lemmas, induction, query checking	2022, Thms. 4 and 6	Proofs, with user axioms assumed (Remark 10).
Equivalence queries	2022, Thm. 5	Uses Lemmas 4, 5, 9, 10, 11 and Theorem 3, stated without proof. Remark 8 points to similar earlier work for Lemma 11. With restrictions, the conference result [18] does not directly establish observational equivalence.
Algorithm to OCaml code	Audited sources	No theorem supplied.

Ivanov and Künnemann report two further confirmed and fixed soundness bugs found through cross-tool analysis [55, p. 12]. There is implementation evidence for both tools, although the different sources cannot support a comparison of their reliability.

The proposed Tamarin repairs for the variable, channel-alias, natural-subterm and equivalence cases remained unmerged on 29 October 2026 [51, 53, 52, 54]. A repair changes what later runs can establish. To find out whether an earlier analysis was affected, its authors still need to check the model, the features it used and the tool version that ran it.

4.3 ProVerif's translation proofs leave gaps

ProVerif translates an input several times before its solver answers the query. Table 2 follows these steps for a model with equations. The 2022 technical report proves much of the later procedure and formalizes optimizations that earlier accounts left undefined. It describes the scope as "the entire syntax and semantics of ProVerif" [19, p. 4]. The calculus it defines is smaller than that description suggests.

In the report's core language, a constructor f has only the identity rule $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$ and rules that propagate failure [19, p. 36]. Equations require more. To express, for example, that two Diffie–Hellman computations give the same shared secret, ProVerif translates equations into extra rewrite rules on constructors [20, p. 34]. Those rules fall outside the 2022 calculus as stated. The 2008 paper introducing this translation proves the needed result for equivalence. For secrecy and correspondence, it says a similar result holds and adds: "These extensions are implemented in ProVerif. We do not detail them further since we focus on equivalences in this paper." [16, p. 17] The audited papers therefore leave a link unproved between models with equations and the 2022 procedure. The proof might extend, but that extension needs to be stated and justified.

The same issue appears in smaller forms elsewhere. The 2016 tutorial points to a similar result in the 2005 secrecy paper, whose calculus has no equations, phases or tables [20, p. 54][1, Thm. 7.2.3]. The 2009 injective-agreement theorems require that "different occurrences of event have different root function symbols" [21, p. 39]. Injective agreement counts distinct matches, so two later events cannot both rely on one earlier event. The paper does not say whether the tool enforces the stated condition. User axioms create a more explicit limit: ProVerif assumes them rather than proving them [19, p. 44]. In each case, applying the theorem takes more than citing its conclusion.

4.4 Tamarin's proof does not cover all its optimizations

Tamarin's theory paper promises "a full proof of its correctness along with proofs of all theorems and assertions in this paper" [57, p. 1]. It proves soundness and completeness of the constraint-reduction rules, treating some cases by analogy. The proof also needs conditions on the equations it reasons with: AC-convergence and the finite variant property (FVP). Roughly, these let the solver reduce terms consistently and describe their possible instances using finitely many variants. The paper cites a result for Diffie–Hellman [28], then asserts the property for its extension with pairing and subterm-convergent operators [57, p. 24].

Work on checking FVP is relevant here [34], and the later Tamarin extension explicitly requires FVP and disjointness from its built-in theories [33, Secs. 2.1, 3]. I treat the 2012 passage as a missing justification in that presentation. It does not show that the combination lacks FVP or that no argument exists elsewhere. Another sentence says normal dependency graphs "allow exactly the same executions as our multiset rewriting semantics" [57, p. 8]; Lemma 1 gives the narrower formal statement, relating observable traces after normalization and modulo associativity and commutativity. These are limits on what the prose has established, rather than counterexamples to the core result.

The connection to the implementation is more consequential. Meier says that the tool uses "an optimized constraint-reduction relation" [47, p. 153], and refers to Schmidt's thesis for the central optimization. Schmidt gives a paragraph of justification and rejects one candidate rule by counterexample, but supplies no proof for the remaining rules [56, p. 78]. The counterexample itself shows why an optimization needs an argument: changing a search rule can change which executions remain possible. A proof of the original rules cannot establish that the optimized ones preserve the answer.

Induction gives both a small omission and a larger unresolved question. Meier's written translation omits the empty-trace base case that the code does check [47, p. 153]. That is a defect in the text. A repair merged on 28 October 2026 instead concerns a supported feature: it forbids the `last` atom in restrictions, citing a corollary in the thesis that covers only formulas without `last`. The maintainer asks, "I wonder how we want to consider restrictions in Corollary 2 anyway." [45] The corollary alone cannot justify the full interaction between restrictions and induction.

4.5 False claims about translations and constraint solving

Some statements need to be changed, rather than merely given a fuller proof. SAPIC+'s optional state optimization is one example. It replaces a "pure" state cell by a private channel, and Lemma 5 says the translation preserves traces. The translation is used for both Tamarin and ProVerif exports [27, p. 37]. Hammann reported this process [40]:

```
new s; insert s, '0'; lookup s as w in event B(w) else event NB()
```

It puts `'0'` in a cell and reads it, so B happens. The cell counts as pure under the published definition, which constrains only inserts: there is one insert outside a lock. The translation turns that insert into an output on a private channel but leaves the bare lookup unchanged. The lookup now sees an empty store and records NB instead. The translation has changed the traces, and the false claim "B never happens" holds afterward.

The proof assumes that "this lookup must have been preceded by a lock" [27, p. 38], though the definition has no such requirement. Its case analysis also omits `delete`; another reported process using deletion breaks the lemma in the other direction [41]. Tamarin 1.12.0 follows a different code path from the paper's translation, but with the optimization enabled it too gets the first process wrong, falsifying both of its correct lemmas. The false printed lemma and the observed tool failure should therefore be distinguished even though they concern the same optimization.

A second example concerns Tamarin's subterm operator <<. The supporting lemma says that "If the constraint system is solved, there is a dependency graph satisfying all constraints" [31, p. 28]. Its proof fills variables with terms containing the required subterms, then re-checks equalities, inequalities, guarded formulas and negative subterm constraints. It does not check whether the attacker can send the new term. Create a fresh secret n and never release it, then accept an attacker input x. The constraint $n \sqsubset x$ can remain in a solved system even though the attacker cannot construct such an input. Tamarin 1.12.0 falsifies "no input contains n" and verifies that an input containing n exists. The repair discussion describes the leftover constraint as "mistaken for a completed witness" [50]. This failure affects models using << and shows how the same bug can produce both a spurious attack and a false executability result.

The 2009 ProVerif paper contains a different kind of error, in its description of moving name restrictions. Remark 2 discusses how this can add arguments to the clause representation of a name. Its full first sentence is: "By moving restrictions downwards in the syntax tree of the process (until the point at which the fresh name is used), one can add more arguments to the pattern that represents the fresh name, when the restriction is moved under an input, replication, or destructor application." [21, p. 22] The remark then says this can improve precision and that "The tool can perform this transformation automatically" [21, p. 22]. The context explains the purpose, but gives no condition permitting a move through replication.

Creating one secret shared by all copies of a process differs from creating a new secret in each copy. With c public and s private, consider

$$(\nu a)\,!\,c(y).\,\mathsf{if}\ y = a\ \mathsf{then}\ \bar{c}\langle s\rangle\ \mathsf{else}\ \bar{c}\langle a\rangle.$$

The process creates a once. An attacker sends another value to one copy, learns a from the else branch, and sends a to a second copy to learn s. Move the restriction inside replication and after the input, and each copy creates a fresh a that cannot equal the value already received. Now s stays secret. So the move is unsafe when the remark is read as a generally applicable preprocessing transformation. ProVerif 2.05 correctly distinguishes the two processes, and its `movenew` option is documented as moving restrictions under inputs [22, Sec. 6.6.2]. This finding concerns the remark's wording; it neither refutes the correspondence theorem nor shows that the tool performs the unsafe move.

Schmidt's thesis has two problems in its printed rule sets. Its extension to multisets and bilinear pairings is described as sound and complete [56, p. 91], but the rule for deconstruction chains starting at a multiset omits a chain consisting of a single edge. Chains are also allowed to end at any premise. Take a node that learns the multiset $A \sharp B$, followed by a node that extracts A:

$$\left\{ \begin{array}{l} l : \mathsf{Out}(A \sharp B) \to \mathsf{K}^{\Downarrow}(A \sharp B), \\ k : \mathsf{K}^{\Downarrow}(A \sharp B) \to \mathsf{K}^{\Downarrow}(A), \\ l \twoheadrightarrow (k, 1). \end{array} \right\}$$

A direct edge gives this constraint system a model, but none of the cases produced by the rule has one. That contradicts Theorem 3.33 in the direction needed to justify a universal "verified" result. Tamarin's implementation tries the direct edge first, so it avoids this printed mistake. The thesis's base rules also omit a rule for negated time ordering, $\neg(i \prec j)$, that the 2012 paper includes. Without it, an impossible system can be called solved, contradicting Theorem 3.36 in the attack-reconstruction direction. This second problem is with the thesis's rules, not the different rule set in the paper.

4.6 Negated equalities and omitted variable renaming

The proof of Theorem 3 in the Tamarin paper dismisses negated equalities because a solved system supposedly cannot contain them [57, p. 40]. The cited rules exclude $\neg(t \approx t)$ but

do not exclude $\neg(x \approx y)$. An injectivity argument may repair the proof, but the paper does not give one. This is a gap in the argument rather than a counterexample to the result.

The 2005 ProVerif translation has a smaller presentation problem [1]. With e public and n and s private, the process

$$(\nu c)(\bar{c}\langle n\rangle \mid c(x).e(x).\bar{e}\langle s\rangle)$$

leaks s. A literal translation that reuses one clause variable for both input binders requires the attacker to send n on e, and so misses the leak. Renaming the second binder fixes it. The 2016 tutorial supplies the fresh-variable convention, so this is no counterexample to the analysis with that convention in place. It belongs with omissions in exposition, not with reproduced false security verdicts.

4.7 The core theorems do not justify every verdict

The audit does not refute either core method. It shows why a citation to the core theorem cannot justify every feature and optimization in the program. Some missing arguments may be supplied, some printed statements need correction, and some implementation bugs return false positive results on small inputs. Those are different findings, but they all bear on how much evidence a user receives with a verdict.

An alleged attack or execution can be checked by replaying it against the submitted model. A universal claim needs an argument that all relevant counterexamples were covered and excluded. ProVerif supplies no certificate for the full procedure, while replaying Tamarin's tree still trusts its solver and the translation of the input. Independent checking would help with these problems. In the next section I assume they have all been solved, so that we can examine what a correct symbolic theorem still leaves to be established about cryptography and code.

5 Model assumptions and the scope of security claims

Suppose every bug in the previous section is fixed, every missing proof is supplied, and an independent checker confirms the result. We now have a theorem about the model. To claim security for the protocol that runs, someone still has to check that the model includes the relevant behavior of the code, that the property says what users need, and that the rules for cryptography are justified by the primitives in use. These are separate jobs, shared with other proof methods. This section addresses what a report should claim, rather than whether a modeling error disqualifies a tool as a prover. A better checker cannot tell us about a parser the model never described, and a stronger secrecy query cannot expose a cryptographic behavior the model leaves out.

5.1 Parsing, state, and behavior omitted from the model

A symbolic model often represents a message with two fields as a pair. A free pairing constructor keeps the fields distinct: equal pairs must have equal components. A program has to encode that pair as bytes. If it simply concatenates variable-length strings, then $(\texttt{ab}, \texttt{c})$ and $(\texttt{a}, \texttt{bc})$ both become `abc`. Suppose the fields carry authorization information and are authenticated as associated data. A valid tag on those bytes authenticates either interpretation. The attacker has not forged anything; the two parties disagree about what the authenticated bytes mean. RFC 5116 explicitly requires an injective encoding of associated-data fields for this reason [46, Sec. 3.3].

The small script accompanying this paper demonstrates the ambiguity with ChaCha20-Poly1305. It verifies the same ciphertext under both pairs of fields because both produce

the same associated data. A symbolic model using a free pair has excluded this behavior before analysis begins. Fixing it requires an encoding that distinguishes the fields and a parser that enforces the distinction. Even a proof that $\mathsf{decode}(\mathsf{encode}(m)) = m$ would only cover honestly encoded messages. An attacker can send other bytes, so the analysis also has to account for everything the parser accepts.

Fresh names make a similar assumption about execution. A fresh symbolic nonce cannot equal another fresh nonce. Code must obtain suitable randomness or keep a counter unique across the key's lifetime. A counter may be shared across threads or devices, and it may need to survive a crash. A model that allocates a nonce and advances a counter in one atomic step does not describe an implementation that can restart between them. RFC 5116 treats coordination between encryptors and persistence across reboots as explicit requirements [46, Secs. 3.1–3.2]. Writing "fresh" in the model doesn't establish either.

The WPA2 attacks known as KRACK show why such details belong in a security argument. The four-way handshake had proofs of key secrecy and session authentication in Protocol Composition Logic, a different verification approach from the tools studied here [42]. KRACK did not need to refute those properties. As Vanhoef and Piessens put it, "the proofs do not model key installation" [59, p. 13]. The protocol's code had to install the key before using it, and reinstalling it could reset the state that prevented nonce reuse. A proof that omitted that operation could not settle whether it was safe.

The environment can matter just as much. Basin, Sasse and Toro-Pozo proved properties of EMV contactless transactions with Tamarin, then found the card brand mixup attack [11, 12]. Their explanation was that "Our previous model of the EMV contactless protocol specifies the terminal-issuer channel in a way that these two parties always agree on the kernel used for online-authorized transactions." [12, p. 12] The attack exploited disagreement about which card brand's kernel ran. The earlier result held under an agreement assumption that the system did not satisfy.

Other omissions are less visible in a message-flow diagram. If all decryption failures have one unobservable outcome, the model cannot distinguish code whose error responses or timings reveal different information. If a secret is one opaque term, its length does not leak unless the model exposes it. Memory safety and foreign-function calls need attention too when the claim extends to their behavior. We do not need to model every implementation detail, but we do need an argument that the details omitted cannot break the claimed property.

For trace properties, that argument has a simple form. Let I be the implementation, M the model, and α a map from every implementation trace to a model trace. Let Φ be the intended property of the implementation and φ the property checked in the model. It is sufficient to show

$$t \in \mathsf{Traces}(I) \implies \alpha(t) \in \mathsf{Traces}(M), \tag{1}$$

$$\alpha(t) \models \varphi \implies t \models \Phi. \tag{2}$$

The first condition says that the model covers every implementation execution. The second says that passing the model's test implies passing the intended one. If every model trace satisfies φ, the two conditions give Φ for every implementation trace. This is elementary, but it identifies what the verdict alone does not supply. Computational security also needs probabilities and resource bounds, while indistinguishability needs a comparison of observations or distributions rather than a property of each trace separately.

There is work that makes these connections explicit. The miTLS implementation covers parsing, fragmentation, errors and application interfaces, with computational assumptions for its primitives; it also says that timing lies outside its formal guarantee [13]. DY* uses symbolic reasoning to verify executable F* protocol code, including message formats, state machines and mutable state [14]. These are substantial improvements over a

handwritten account of message flow. Their guarantees still depend on the cryptographic interfaces and observations they formalize. Reaching executable code and justifying a cryptographic abstraction solve different parts of the problem.

5.2 Secrecy, indistinguishability, and authentication

A faithful model can still be asked a question weaker than the claim in the report. ProVerif distinguishes reachability secrecy from strong secrecy: whether the attacker can derive a value, and whether changes in that value can be observed [22, Sec. 4.3.2]. Both may be described informally as secrecy, but they promise different things. Suppose a process publishes $h(s)$, where s is a private name and h is a public constructor with no inverse. ProVerif 2.05 correctly reports that the attacker cannot derive s.

Now replace s by one of two distinct public candidates, a and b. An attacker can compare the output with $h(b)$ and tell which candidate was used. This does not recover the original private name; it shows that the output changes observably with the secret. The accompanying equivalence model reports "cannot be proved" and supplies that distinguishing trace. The verdict by itself is inconclusive, but the equality test establishes the difference. ProVerif supports the stronger question. The mistake would be to report an answer to the weaker one as if it settled both.

Restrictions and axioms need the same care. A property saying that every acceptance has a matching send is vacuously true if the process never accepts anything. Checking that an acceptance can occur helps, but it does not justify every restriction used to exclude other executions. A restriction can leave honest sessions intact and remove the only attack. ProVerif also assumes user axioms rather than proving them [19, p. 44]. A report therefore needs to tell the reader which facts the tool established and which facts the analyst supplied.

Even the parameters of an authentication event change what the theorem says. Agreement on a key cannot establish agreement on a peer identity that the event never records. The analyst has to choose and explain those parameters, just as they have to choose what secrecy means.

5.3 Computational guarantees require justified assumptions

A computational proof describes what an attacker can see and do in an experiment on bitstrings. It then shows how an efficient attacker against the protocol would give an efficient attacker against an assumed property of a primitive. The reduction states how much time and advantage are lost. Tools such as CryptoVerif can organize this argument as a sequence of games and compute concrete bounds [25, 23]. The proof still uses a model, but its assumptions have conditions that the argument must meet.

Symbolic reasoning can contribute to such an argument, but a decryption equation is not itself a computational security assumption. The connection needs a justification. ProVerif's manual states its scope explicitly: "we make no computational soundness claims: ProVerif relies on the symbolic, Dolev-Yao model of cryptography; its results do not apply to the computational model, at least not directly." [22, p. 57] Bana and Comon-Lundh likewise say that a positive result must be read as security in the model, with no guarantee that the abstractions suit the implementation [7].

Soundness theorems and richer primitive models. Computational soundness theorems can connect symbolic results to cryptographic ones, provided their hypotheses hold. Abadi and Rogaway's foundational result, for example, excludes certain encryption cycles [2]. Cortier, Kremer and Warinschi's survey explains the strong primitive assumptions many such results need [29]. Other results show limits: Backes and Pfitzmann restrict how far strong soundness results extend to XOR, while work on idealized hashes gives limits

in the standard model and a positive result in the random-oracle model [4, 5]. These are results about particular definitions and assumptions. Citing computational soundness in general does not tell us which theorem applies to the model at hand.

Richer symbolic models also help. *Seems Legit* models signature behaviors permitted by standard unforgeability definitions that ideal signature abstractions leave out [43]. *Hash Gone Bad* adds finer hash threat models to ProVerif and Tamarin so that they can find attacks involving hash weaknesses [26]. DY* addresses executable protocol code, as discussed above. Each improves what can be analyzed, and each requires work beyond running a model with the simpler abstraction. A result obtained before that work does not acquire the improved coverage just because the same tool can now express it.

CryptoVerif and Squirrel show that mechanized reasoning can have computational meaning [25, 6]. Their checking code still needs to be correct, and their models still need to describe the relevant implementation behavior. A report should state which of these connections has been established and under which assumptions. The next section proposes a way to do that.

6 Report the model, assumptions, and evidence

ProVerif and Tamarin have helped find real attacks. That record is a good reason to use them, and it gives a practical starting point for reporting what they do. An attack trace tells a reader which steps to examine. The reader can replay them against the model, then try them against the intended system. Either check can fail, but when both succeed we have evidence of a problem that goes beyond the tool's announcement.

Bhargavan et al. used ProVerif to find flaws in the first version of Signal's PQXDH, including a re-encapsulation attack, and helped incorporate changes into the second version [15]. Basin, Sasse and Toro-Pozo used Tamarin to find the EMV card brand mixup attack and report that Mastercard deployed defenses [12]. Cremers et al. found an attack on a client-authentication mechanism proposed for TLS 1.3 draft-10 [30]. These are concrete contributions to protocol design. They also show what a reader can learn from a trace: what the attacker sends, what each party accepts, and where the intended guarantee fails.

The PQXDH analysis illustrates how several kinds of evidence can work together. A failed CryptoVerif proof suggested a public-key confusion problem, but the authors warned that "failing to do a proof does not imply that this proof cannot be made" [15, p. 9]. ProVerif then supplied a symbolic attack trace. The failed proof gave the researchers a reason to investigate; the trace showed how the modeled protocol could go wrong. A report that distinguishes those steps is more useful than one that calls the whole exercise a verification.

A positive result also deserves a precise description. For universal queries, I would distinguish "attack found", "no attack found; search completed", and "inconclusive". The second requires a positive verdict, not merely a process that stopped. A terminating "cannot be proved" belongs with timeouts and resource exhaustion in the third category. If the result covers unbounded symbolic sessions, say so. For an existence query, state which execution is claimed and supply its trace where possible. For an alleged attack, say whether anyone replayed it against the model and tested it against the system.

The rest of the report should answer five practical questions. I propose the following fields so that a reader doesn't have to reconstruct the answers from a paper's methods section, code repository and informal claims:

1. *What was analyzed?* Identify the model by revision or hash, give the exact query, and explain what it means. State the primitive equations and what the attacker can observe.

2. *How was it run?* Give the tool version, command and options. List restrictions, axioms, helper lemmas and whether those lemmas were proved, along with any bounds on sessions or terms.
3. *What evidence was produced?* Record the verdict, whether the search finished, the output and any proof object or witness. Say how it was checked and whether an execution was independently replayed.
4. *What connects it to code?* Name the code revision and the argument that the model covers it. If that connection is an informal judgment or has not been checked, say so.
5. *What connects it to cryptography?* Name the reduction or transfer theorem, its assumptions and its security bound. If no such argument was supplied, record that too.

For a computational security claim, the report needs a computational argument, whether written by hand or developed with a tool such as CryptoVerif or Squirrel. For a symbolic theorem, a checkable symbolic argument may be enough, provided the claim stays within it; scyther-proof and DyLean show ways to obtain such arguments [48, 60]. Symbolic analysis can help with either effort by finding counterexamples, testing assumptions and identifying where a proof fails. I want reports to preserve those benefits while making it clear which further work has actually been done.

7 Call the tools symbolic protocol analyzers

ProVerif and Tamarin do useful work, but a successful run leaves more to establish than the word "prover" suggests. The audit found bugs that accept false claims about the submitted model, missing arguments between the published theory and the procedure in use, and printed statements that need correction. These findings have different consequences, and none establishes that every result is wrong. They do establish that a citation to the core soundness theorem is not enough to justify the tool's answer.

The practical demand is to make the evidence easier to check. Show which argument justifies the procedure that ran, which parts of that procedure were checked independently, and what the result actually says. Model authors have their own responsibility to choose suitable assumptions and questions, just as they do with any other proof method. That responsibility does not excuse a wrong answer about the model they supplied. I would call these tools symbolic protocol analyzers and reserve claims of proof for the arguments a report actually makes available. The value of the analysis would remain; the reader would have a clearer account of what supports it.

Acknowledgments. I thank Mario Raso for noticing that two counterexamples in Section 4 did not say which names were private.

Disclosure. I develop Verifpal, a bounded symbolic protocol analyzer, and coauthored the ProScript/Signal analysis (EuroS&P 2017) and Noise Explorer (EuroS&P 2019), both of which used ProVerif. For the reasons outlined in this paper, Verifpal has never called itself a prover. My own ProVerif analyses should be reported under the same rules.

References

[1] M. Abadi, B. Blanchet. *Analyzing Security Protocols with Secrecy Types and Logic Programs.* Journal of the ACM 52(1):102–146, 2005.

[2] M. Abadi, P. Rogaway. *Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption).* Journal of Cryptology 15(2):103–127, 2002.

[3] K. Appel, W. Haken. *Every Planar Map Is Four Colorable.* Bulletin of the American Mathematical Society 82(5):711–712, 1976.

[4] M. Backes, B. Pfitzmann. *Limits of the Cryptographic Realization of Dolev-Yao-Style XOR.* ESORICS 2005, LNCS 3679, pp. 178–196.

[5] M. Backes, B. Pfitzmann, M. Waidner. *Limits of the BRSIM/UC Soundness of Dolev-Yao Models with Hashes.* ESORICS 2006, LNCS 4189, pp. 404–423.

[6] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, S. Moreau. *An Interactive Prover for Protocol Verification in the Computational Model.* IEEE S&P 2021, pp. 537–554.

[7] G. Bana, H. Comon-Lundh. *Towards Unconditional Soundness: Computationally Complete Symbolic Attacker.* POST 2012, LNCS 7215, pp. 189–208.

[8] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, B. Parno. *SoK: Computer-Aided Cryptography.* IEEE S&P 2021, pp. 777–795. https://bblanche.gitlabpages.inria.fr/publications/BarbosaetalOakland21.pdf.

[9] H. Barendregt, F. Wiedijk. *The Challenge of Computer Mathematics.* Philosophical Transactions of the Royal Society A 363(1835):2351–2375, 2005. Page numbers refer to the authors' version.

[10] D. Basin, J. Dreier, R. Sasse. *Automated Symbolic Proofs of Observational Equivalence.* ACM CCS 2015, pp. 1144–1155.

[11] D. Basin, R. Sasse, J. Toro-Pozo. *The EMV Standard: Break, Fix, Verify.* IEEE S&P 2021, pp. 1766–1781.

[12] D. Basin, R. Sasse, J. Toro-Pozo. *Card Brand Mixup Attack: Bypassing the PIN in non-Visa Cards by Using Them for Visa Transactions.* USENIX Security 2021, pp. 179–194.

[13] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub. *Implementing TLS with Verified Cryptographic Security.* IEEE S&P 2013, pp. 445–459. https://doi.org/10.1109/SP.2013.37.

[14] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, R. Küsters, G. Schmitz, T. Würtele. *DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code.* IEEE EuroS&P 2021, pp. 523–542. https://doi.org/10.1109/EuroSP51992.2021.00042.

[15] K. Bhargavan, C. Jacomme, F. Kiefer, R. Schmidt. *Formal Verification of the PQXDH Post-Quantum Key Agreement Protocol for End-to-End Secure Messaging.* USENIX Security 2024, pp. 469–486.

[16] B. Blanchet, M. Abadi, C. Fournet. *Automated Verification of Selected Equivalences for Security Protocols.* Journal of Logic and Algebraic Programming 75(1):3–51, 2008.

[17] B. Blanchet. *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules.* 14th IEEE Computer Security Foundations Workshop (CSFW-14), pp. 82–96, 2001.

[18] B. Blanchet, V. Cheval, V. Cortier. *ProVerif with Lemmas, Induction, Fast Subsumption, and Much More.* IEEE Symposium on Security and Privacy (S&P 2022), pp. 205–222, 2022.

[19] B. Blanchet, V. Cheval, V. Cortier. *ProVerif with Lemmas, Induction, Fast Subsumption, and Much More.* Long version with proofs, 156 pp. https://bblanche.gitlabpages.inria.fr/proverif/snp22/technical_report.pdf, accessed 28 October 2026.

[20] B. Blanchet. *Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif.* Foundations and Trends in Privacy and Security 1(1–2):1–135, 2016.

[21] B. Blanchet. *Automatic Verification of Correspondences for Security Protocols.* Journal of Computer Security 17(4):363–434, 2009. Version with proofs: arXiv:0802.3444v1.

[22] B. Blanchet, B. Smyth, V. Cheval, M. Sylvestre. *ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.* 17 October 2023.

[23] B. Blanchet, D. Pointcheval. *Automated Security Proofs with Sequences of Games.* CRYPTO 2006, LNCS 4117, pp. 537–554. https://doi.org/10.1007/11818175_32.

[24] B. Blanchet et al. *ProVerif release history.* `CHANGES` file distributed with ProVerif 2.05, checked against the official release archive on 29 October 2026. https://proverif.inria.fr/proverif2.05.tar.gz.

[25] B. Blanchet. *A Computationally Sound Mechanized Prover for Security Protocols.* IEEE Transactions on Dependable and Secure Computing 5(4):193–207, 2008.

[26] V. Cheval, C. Cremers, A. Dax, L. Hirschi, C. Jacomme, S. Kremer. *Hash Gone Bad: Automated Discovery of Protocol Attacks that Exploit Hash Function Weaknesses.* USENIX Security 2023, pp. 5899–5916. https://www.usenix.org/conference/usenixsecurity23/presentation/cheval.

[27] V. Cheval, C. Jacomme, S. Kremer, R. Künnemann. *Sapic+: Protocol Verifiers of the World, Unite!* USENIX Security 2022, pp. 3935–3952. Technical report with proofs: HAL hal-03693843v1.

[28] H. Comon-Lundh, S. Delaune. *The Finite Variant Property: How to Get Rid of Some Algebraic Properties.* Term Rewriting and Applications (RTA 2005), LNCS 3467, pp. 294–307, 2005.

[29] V. Cortier, S. Kremer, B. Warinschi. *A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems.* Journal of Automated Reasoning 46(3–4):225–259, 2011.

[30] C. Cremers, M. Horvat, S. Scott, T. van der Merwe. *Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication.* IEEE S&P 2016, pp. 470–485.

[31] C. Cremers, C. Jacomme, P. Lukert. *Subterm-Based Proof Techniques for Improving the Automation and Scope of Security Protocol Analysis.* Cryptology ePrint Archive, Paper 2022/1130, version of 30 August 2022.

[32] J. P. Degabriele, K. G. Paterson, G. J. Watson. *Provable Security in the Real World.* IEEE Security & Privacy 9(3):33–41, 2011.

[33] J. Dreier, C. Duménil, S. Kremer, R. Sasse. *Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols.* POST 2017, LNCS 10204, pp. 117–140. Extended version with proofs: https://people.inf.ethz.ch/rsasse/pub/BeyondSubterm-POST17-techreport.pdf.

[34] S. Escobar, J. Meseguer, R. Sasse. *Effectively Checking or Disproving the Finite Variant Property.* RTA 2008, LNCS 5117, pp. 79–93. Author draft titled *Effectively Checking the Finite Variant Property*: https://personales.upv.es/sanesro/papers/rta08-2.pdf.

[35] G. Gonthier. *Formal Proof—The Four-Color Theorem.* Notices of the AMS 55(11):1382–1393, 2008.

[36] J. Goubault-Larrecq. *Towards Producing Formally Checkable Security Proofs, Automatically.* IEEE CSF 2008, pp. 224–238. Page numbers refer to Research Report LSV-08-15, ENS Cachan, 2008.

[37] T. Hales, M. Adams, G. Bauer, et al. *A Formal Proof of the Kepler Conjecture.* Forum of Mathematics, Pi 5:e2, 2017.

[38] S. Hammann (GitHub user SvenHammann90). *SAPIC: builtin equation variables `x.1`/`x.2` collide with renamed process variables in destructor `let`.* Tamarin issue #936, opened 15 September 2026, open on 28 October 2026. https://github.com/tamarin-prover/tamarin-prover/issues/936.

[39] S. Hammann (GitHub user SvenHammann90). *SAPIC: secret-channel analysis runs before `let` substitution.* Tamarin issue #938, opened 15 September 2026, open on 29 October 2026. https://github.com/tamarin-prover/tamarin-prover/issues/938.

[40] S. Hammann (GitHub user SvenHammann90). *SAPIC state optimisation: `lookup` on a "pure" cell outside a `lock` gets an unrestricted `IsIn`.* Tamarin issue #941, opened 15 September 2026, open on 28 October 2026. https://github.com/tamarin-prover/tamarin-prover/issues/941.

[41] S. Hammann (GitHub user SvenHammann90). *SAPIC state optimisation: successive `inserts` do not overwrite, `delete` is ignored, and two `inserts` in one locked section deadlock.* Tamarin issue #942, opened 15 September 2026, open on 28 October 2026. https://github.com/tamarin-prover/tamarin-prover/issues/942.

[42] C. He, M. Sundararajan, A. Datta, A. Derek, J. C. Mitchell. *A Modular Correctness Proof of IEEE 802.11i and TLS.* ACM CCS 2005, pp. 2–15.

[43] D. Jackson, C. Cremers, K. Cohn-Gordon, R. Sasse. *Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures.* ACM CCS 2019, pp. 2165–2180. https://doi.org/10.1145/3319535.3339813.

[44] N. Koblitz, A. J. Menezes. *Another Look at "Provable Security".* Journal of Cryptology 20(1):3–37, 2007. https://doi.org/10.1007/s00145-005-0432-z.

[45] F. Linker (GitHub user felixlinker). *Do not use last atoms in formulas.* Tamarin pull request #915, opened 12 August 2026, merged 28 October 2026. https://github.com/tamarin-prover/tamarin-prover/pull/915.

[46] D. McGrew. *An Interface and Algorithms for Authenticated Encryption.* RFC 5116, January 2008. https://www.rfc-editor.org/rfc/rfc5116.

[47] S. Meier. *Advancing Automated Security Protocol Verification.* PhD thesis, ETH Zurich, 2013. DOI 10.3929/ethz-a-009790675.

[48] S. Meier, C. Cremers, D. Basin. *Strong Invariants for the Efficient Construction of Machine-Checked Protocol Security Proofs.* IEEE CSF 2010, pp. 231–245.

[49] S. Meier, B. Schmidt, C. Cremers, D. Basin. *The TAMARIN Prover for the Symbolic Analysis of Security Protocols.* CAV 2013, LNCS 8044, pp. 696–701.

[50] K. Milner (GitHub user kmilner). Comment on Tamarin pull request #948, *Soundness/input validation/general correctness fixes for regular proving*, 17 September 2026. https://github.com/tamarin-prover/tamarin-prover/pull/948#issuecomment-5710868426.

[51] K. Milner (GitHub user kmilner). *[SAPIC Step 1] Fix SAPIC destructors.* Tamarin pull request #957, opened 26 September 2026, open on 29 October 2026. https://github.com/tamarin-prover/tamarin-prover/pull/957.

[52] K. Milner (GitHub user kmilner). *[MSR Step 2] Proof correctness.* Tamarin pull request #958, opened 26 September 2026, open on 29 October 2026. https://github.com/tamarin-prover/tamarin-prover/pull/958.

[53] K. Milner (GitHub user kmilner). *[SAPIC Step 2] Fix some incorrect SAPIC optimisations.* Tamarin pull request #959, opened 1 October 2026, open on 29 October 2026. https://github.com/tamarin-prover/tamarin-prover/pull/959.

[54] K. Milner (GitHub user kmilner). *[Diff Step 1] Diff Mode general fixes.* Tamarin pull request #964, opened 2 October 2026, open on 29 October 2026. https://github.com/tamarin-prover/tamarin-prover/pull/964.

[55] K. Morio, Y. Ivanov, R. Künnemann. *A Sound Translation from Tamarin to ProVerif: Enabling Comparative Analysis.* arXiv:2608.06315, 2026. Full version of an ACM CCS 2026 paper.

[56] B. Schmidt. *Formal Analysis of Key Exchange Protocols and Physical Protocols.* PhD thesis, ETH Zurich, 2012.

[57] B. Schmidt, S. Meier, C. Cremers, D. Basin. *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties.* IEEE Computer Security Foundations Symposium (CSF 2012), pp. 78–94. Extended version 1, 24 April 2012, with proofs. https://infsec.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/dh_tamarin_extended_v1.pdf.

[58] The Tamarin Team. *Tamarin-Prover Manual: Security Protocol Analysis in the Symbolic Model.* Version of 7 March 2026.

[59] M. Vanhoef, F. Piessens. *Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2.* ACM CCS 2017, pp. 1313–1328.

[60] T. Wallez, C. Cremers. *Bob DyLean: A Framework for the Symbolic Analysis of Cryptographic Protocols in Lean.* Cryptology ePrint Archive, Paper 2026/1493.