BP: Formal Proofs, the Fine Print and Side Effects

Toby Murray†* P.C. van Oorschot‡†
†University of Melbourne *Data61, Australia ‡Carleton University, Canada

***Abstract*—Given recent high-profile successes in formal verification of security-related properties (e.g., for seL4), and the rising popularity of applying formal methods to cryptographic libraries and security protocols like TLS, we revisit the meaning of security-related proofs about software. We re-examine old issues, and identify new questions that have escaped scrutiny in the formal methods literature. We consider what value proofs about software systems deliver to end-users (e.g., in terms of net assurance benefits), and at what cost in terms of side effects (such as changes made to software to facilitate the proofs, and assumption-related deployment restrictions imposed on software if these proofs are to remain valid in operation). We consider in detail, for the first time to our knowledge, possible relationships between proofs and side effects. To make our discussion concrete, we draw on tangible examples, experience, and the literature.**

I. Introduction

Proofs have been used as a means to aid the development of secure systems, especially cryptographic protocols and critical software like separation kernels, for at least the past five decades [1, Chapter 5]. During this time there has been much debate about proof's role in developing secure software and systems, including its practicality for mainstream software [2], relevance for reasoning about the real-world security of cryptographic protocols [3], and so on. Yet we live in a time in which the popularity of proofs has increased sharply, perhaps thanks to recent breakthroughs in mechanised reasoning [4] [5] and a string of high-profile success stories.

Over the past decade, marked by the seminal functional correctness verification of the seL4 microkernel [6], there has been a rash of large-scale software verification projects that have resulted in the construction of software systems whose security assurance is backed by (machine-checked) proofs about the software's behaviour. Examples include formally verified instant messenger kernels [7], TLS stacks [8], browser kernels [9], conference management systems [10], social media platforms [11], hardware architectures [12], multicore kernels [13], secure application stacks [14], distributed systems [15] [16], and crypto algorithm implementations [17] amongst others. In light of this trend, it is timely to critically re-examine the role of proof for developing secure software.

Despite, or because of, its increasing popularity, we contend that as an assurance mechanism proof remains poorly understood especially outside of formal verification practitioners and researchers. Yet, as we explain in Section II, a scientific understanding of precisely what assurance benefits a proof does, and does not, deliver is vital to ensure we reap maximum benefit from this rapidly evolving technology.

Indeed, proof surely delivers many benefits, and examples abound of defects and vulnerabilities found during the process of proving (a property about) a system that led to improvements in the system's security that might have been unlikely otherwise. Yet, as we identify in Section III, proof also brings with it its own *side effect*s (code changes and deployment constraints), not all of which may be positive for system security and some of which might even be negative (aside from on performance). So far we have neither a systematic understanding of the relationship between these benefits and side effects, nor general rules of thumb for weighing one against another when deciding whether to use proof as a means of assurance. Our hope is to stimulate exploration of these important issues understudied to date.

One especially acute side effect, present in past verification projects, is restrictions on system deployment or functionality to meet rigid proof assumptions. As Section IV explains, this side effect makes (boolean logic) proofs an especially brittle form of assurance, and further complicates the process of judging their value *a priori*. Section V briefly discusses further literature. Section VI summarizes a few questions from our analysis, to better understand the role of proofs in secure software development. Rather than seed doubt on the enterprise of proof and formal verification, we aim to stimulate interest and further exploration of issues raised herein, and believe this may significantly impact future directions of formal methods in security and software verification.

II. The meaning and value of a "Proof"

Few assurance technologies have more mystique, and more controversy [18], [19] than formal proof. In this section, we argue that there is both confusion, and wide variety of interpretations by different stakeholders and communities, about the value and role of formal proofs in the construction of secure systems. While our primary focus is on proofs as related to formal verification, much of the discussion is relevant to security-related proofs in general.

A. *Definition of a Proof (and of Security)*

We begin by setting the terms of the debate. Fix an arbitrary *system* whose *security* is of interest. We leave the term "security" largely undefined (entire papers could pursue this), but note for later use three definitions that one might choose:

1) the system's ability to withstand the actual attacks carried out over a fixed time period;
2) the system's ability to withstand all possible attacks; or
3) its ability to withstand all attacks feasible for attackers with specified computational and observational powers.

Some of these are better suited to our understanding of proofs than others; not all communities will agree on which definition is most appropriate, but many academics gravitate to 3).

By a *proof* we mean a mechanised, logically sound deductive argument, applied to a *formal model* of the (behaviour of) the system, that establishes one or more *properties* of the model. This definition includes proofs over highly detailed models, such as those that reason over the formal semantics of compiled, binary programs [20] [21] in the underlying instruction set architecture (ISA) [22] since even a formal ISA semantics is an idealised model of reality as aptly demonstrated by row-hammer attacks [23] or CPU errata.

Returning to security, it should be evident that a sound definition of security requires a specification of the adversary. This may be separate but closely related to, or an actual part of, the formal model. Thus any security proof results are also subject to the assumption that the adversary in reality adheres to the formal model. A recent exemplar of this viewpoint comes from a 2017 NSF workshop report [24] on formal methods for security (emphasis in original):

> By explicitly modeling the computer system and the abilities of adversaries, formal methods can prove that the computer system is secure against *all* possible attacks (up to modeling assumptions).

The caveat "up to modeling assumptions" implies that "*all* possible attacks" should be construed as restricted to those that conform to the formal model, and "secure" applies to the real system only when its behaviour accords with the formal model. Of the three definitions for "security", the third appears intended here: "secure" meaning the system's ability to withstand an attacker with some fixed set of computational and observational powers, captured in the formal model. This is in contrast to the meaning of security we posit is understood by non-experts: secure against all possible attacks.

Note that proofs alone cannot establish properties of the deployed system: by definition the proof applies *only* to the formal model of the system. Regardless of our definition of security and what constitutes the system in whose security we are interested, the security of the system is necessarily some property of the real world, separate from the formal model.

We sharpen this point with reference to discussion [25] on how *inductive* and *deductive* statements differ. An inductive statement is one relating to the empirical world, based on real-world observations and inferences from them. A deductive statement, derived from axioms and logical rules, relates to abstractions and models.

Proofs are deductive statements, while claims of security for real systems are inductive. Each is a different kind of statement. It has long been accepted that deductive claims offer no guarantees about the real world. It should not be surprising that proofs alone cannot establish claims of real-world security. Proofs cannot even be refuted by real-world observations since any sound proof cannot be refuted by definition. For instance, demonstrating a row-hammer attack against a system that has been formally verified does not invalidate the deductive conclusion that the *formal model* of the system satisfied a *formal* security property, yet clearly has a bearing on the perceived security of the system in the real world. The issue here is that the formal model on which the proof rests implicitly assumes that each memory cell can be modified only by writes to that cell, an assumption row-hammer attacks [23] violate. The validity of such assumptions is a critical aspect of *satisfaction arguments* [26], [27].

This of course does not prevent inductive and deductive reasoning from interacting. Indeed, many reasonable inductive claims are based on deductive reasoning, and observations of the real world that confirm or refute those inductive claims can help to improve the formal models on which further deductive reasoning can be performed.

Example 1: Using the recent Spectre attack [28] as an example, consider the empirical, i.e., inductive claim (assuming a set associative data cache) and C code snippet of Figure 1. One could build a formal model of the behaviour of this code with respect to the data cache and prove that in the formal model the only data cache sets that can be modified are those that can be occupied by the mentioned program variables. However, as of 3 Jan. 2018, one would be unwise to make the empirical claim from Figure 1 based on this deductive reasoning. We now know that in many modern processors that speculative execution can cause modifications to other cache sets (notably those corresponding to `a[i]` for values of `i` $\geq$ `ARRAY_LEN`).

> The code below when run on modern x86 CPUs, can cause modifications only within those data cache sets that can be occupied by the physical memory corresponding to the program variables `i`, `r` and the array `a`, whose length is `ARRAY_LEN`.

```
if (i < ARRAY_LEN){
  r = a[i];         }
```

Fig. 1. An empirical claim and related Spectre-like vulnerable code.

Thus useful deductive reasoning to support the empirical claim of Figure 1 should account for the possibility of cache modification during speculative execution. Yet even if it does, there's nothing to guarantee that there does not exist some other exotic feature of contemporary CPU microarchitecture that we need to incorporate into our formal models next year. *(End of Example 1)*

B. Value of a Proof

What, then, is the value of a formal proof if it cannot establish properties of the real system? We consider a number of different perspectives. Our position is that each offers useful insights and that proofs serve a number of different purposes, providing a range of benefits—and, as we explore later in Section III, also having certain side effects.

1) Proofs as (Qualified) Guarantees: Perhaps the most common interpretation of the meaning of a proof is that it provides *guarantees* about a system, or more precisely as experts recognize, a system model. In the context of security, the kind of guarantees one is interested in is the absence of vulnerability to (specifically identified) real-world attacks.

This perspective is perhaps strongest *outside* of the formal methods community. As of 19 Feb. 2018, Google records over 8,300 pages that mention "seL4" alongside terms such as "unhackable", "invulnerable", "hack-proof", "bug free" or "zero bugs"—despite the second-top Google result for "seL4 proofs" being a FAQ page [29] carefully explaining why the seL4 proofs make no such guarantees. This popular interpretation is not unfair, as historically the words "proof" and "theorem" have implied 100% certainty [30]. The confusion is: certainty about what? When non-experts hear that a system has been "proved secure", many assume this provides guarantees against all possible attacks. Of the three definitions for "security" given in Section II-A, this corresponds to the second.

Within the formal methods community—i.e., those practising formal verification—things are more nuanced. The word "guarantee" appears often in literature on formal proofs of security. For instance, proofs of noninterference for the seL4 kernel were described in a single paper [31] as, on the one hand, "guarantees on information flow provided by a strong machine-checked theorem" and, on the other, "not an iron-clad [security] statement". This apparent contradiction—frequent in the literature, and not to be discounted as a source of confusion—is resolved by observing that a proof provides guarantees subject to the accuracy of the model and proof assumptions, i.e., provides guarantees about the real world when, and only when, the formal model of the system matches the system's real-world behaviour with respect to the property being proven. We identify several categories of assumptions later, in Section III-B.

This view of proofs as guarantees predicated on assumptions is present in much recent work on formally verified software, which often goes to great lengths to carefully enumerate the assumptions on which their proofs depend. Among others, the seL4 project is a good example here [6], [31], [32].

Of course a proof cannot provide guarantees above the formal theorem that it establishes. Even for formal methods *experts*, determining the guarantees a proof implies is nontrivial. Benjamin Pierce—a leading researcher in the field—reported [33] that understanding the main seL4 functional correctness theorem [6] and the "claim it was making about the system," deeply enough to prepare two lectures about them in a graduate seminar, required about a person-week of effort.

The formal methods community is well aware that their proofs involve many assumptions, and pertain to only a *model* of a system operating in an environment against an adversary with fixed abilities. Whether non-experts who read their proofs, papers or abstracts thereof, are equally aware, is a separate question. We note that, given the clear popular interpretation of proofs as inviolable guarantees, lists of assumptions may, in practice, be akin to the "fine print" of legal agreements. Just as an insurance policy may fail to cover you in the event of a disaster because you didn't read the fine print, a proof of security offers little help against a security breach if you fail to carefully ensure that its assumptions and model match reality. Whether or not real systems conform to system models, and real-world attackers conform to attacker specifications, is—perhaps surprisingly—beyond the scope of a proof itself. Of course, non-experts are in no position to verify the relevant assumptions, or compliance with models; and when the assumptions are not even written down, even experts are unable, or otherwise fail to see it as their responsibility.

We use an example to highlight the difficulty of discerning implicit assumptions buried deep in a formal model, and then determining their impact on a formal proof when these assumptions turn out not to match reality.

Example 2: Khakpour et al. [34] prove a series of isolation properties for the ARMv7 instruction set architecture (ISA). Amongst other things, these proofs are designed to establish that the execution of user mode instructions in one process does not leak information to some other process whose address space is separate from the first.

The formal model for the behaviour of each user mode instruction in the ISA includes only their effects on the registers and physical memory, and a few other pieces of ISA state such as exclusive monitor state. If the second isolated process is an attacker trying to infer information from the first process, we must therefore assume that the attacker cannot observe anything besides these, and so for instance cannot measure time. Thus, like almost all large-scale information flow proofs, their proof targets the absence of storage channels but not timing channels [35], and is most appropriate to a definition of security against an attacker with specified capabilities.

Their proofs provide guarantees subject to the formal ISA model matching reality (with respect to the effects of user mode instructions on the ISA state as captured in the formal model). The formal ISA model of Khakpour et al. [34] was derived from that of Fox and Myreen [22], who performed extensive empirical validation of their model to check that it matches the behaviour of a number of ARM processors.

Yet the formal ISA model has several "blind spots"—places in which it is unable to precisely specify the behaviour of the ISA, because no such specification exists. In particular, the ARM reference manual [36] defines a number of cases in which the behaviour of various instructions, under certain conditions, is "UNPREDICTABLE", or the values of certain results are "UNKNOWN". Each of these allows implementors implementation freedom in addressing corner cases. The manual defines UNPREDICTABLE behaviour as follows [36].

> UNPREDICTABLE: Means the behavior cannot be relied upon. UNPREDICTABLE behavior must not perform any function that cannot be performed at the current or a lower level of privilege using instructions that are not UNPREDICTABLE.
> UNPREDICTABLE behavior must not be documented or promoted as having a defined effect.

It defines an UNKNOWN value as follows [36].

> An UNKNOWN value does not contain valid data, and can vary from moment to moment, instruction to instruction, and implementation to implementation.

An UNKNOWN value must not return information that cannot be accessed at the current or a lower level of privilege using instructions that are not UNPREDICTABLE and do not return UNKNOWN values. An UNKNOWN value must not be documented or promoted as having a defined value or effect.

While likely very highly trustworthy for well-defined parts of the ISA, the formal model [34] almost certainly does *not* match the behaviour of any real world ARM processor for UNPREDICTABLE behaviour and UNKNOWN values. This is because the formal model captures the UNPREDICTABLE behaviour akin to an error condition (preventing it from having *any* specified behaviour), and UNKNOWN as always producing an unknown but *fixed* value (namely the special value ARB, a universal constant defined in the logic of the HOL4 prover [37] in which they carried out their proofs). Determining that the model deviates from reality in this way naturally requires one to very carefully examine the formal definitions of the ISA model, which comprise around 7,000 lines of code in the input language of the HOL4 prover (including whitespace and comments).

It might have been more accurate to capture both kinds of behaviour using nondeterminism. So in some sense the formal model upon which these isolation proofs rests is not ideal. Yet, we are inclined to agree with Khakpour et al. [34] that these discrepancies are unlikely to impact the usefulness of their proofs. The manual [36] makes it clear that UNKNOWN values shouldn't leak information that can't otherwise be obtained. While the manual is somewhat ambiguous on how UNPREDICTABLE behaviour relates to information leakage, the intent seems clear that the most likely impact of UNPREDICTABLE behaviour is that it would not leak sensitive information in conforming implementations. Under these assumptions the Khakpour model [34] seems reasonable for the purpose of proving their isolation properties.

This highlights the difficulty of validating implicit assumptions buried deep in a formal model, and makes clear that when considering whether a formal model matches reality one must do so with respect to the property being proved. While the discrepancies above are likely a non-issue (see above) for the properties Khakpour proves, consider this pathological case: trying to prove that a particular instruction always produces the *same* UNKNOWN value as its result. The formal model would almost certainly allow one to prove that this property holds, but clearly that proof should not be relied on as evidence for that claim in reality.

(End of Example 2)

All proofs have assumptions. Yet, as noted by DeMillo, Lipton and Perlis in their infamous critique of formal methods [2], what sets formal verification of software apart from proofs in pure mathematics, say, is their sheer volume of assumptions. While we expect mathematical theorems to provide 100% guarantees (about mathematical objects), we should not expect formal verification to do the same for guarantees about the security of real-world systems. This highlights the potential for misunderstanding when talking about "proofs" in the context of formal verification, a problem noted also by DeMillo et al. [2]. To put it another way: when was the last time that anybody described the safety of a bridge as having been "proved" or "formally verified"? Yet civil engineers use mathematics to inform understanding about the properties of bridges in much the same way that formal verification practitioners use logic to do the same for software. The use of these kinds of terms, in the context of formal methods for software engineering, is at odds with other engineering disciplines.

Terminology aside, from this viewpoint of *proofs as qualified guarantees* has arisen one popular conclusion about the value of formal proofs for security: they allow a security evaluator, when assessing the security of the system, to concentrate their efforts on validating the accuracy of the proof's assumptions, enabling them to ignore large parts of the system's implementation. To quote Klein's 2009 explanation of the seL4 functional correctness proofs to a non-formal methods audience [38]:

The key condition in all this is *if the assumptions above are true*. To attack any of these properties, this is where you would have to look. What the proof really does is take 7,500 lines of C code out of the equation and reduce possible attacks and the human analysis necessary to guard against them to the remaining bits. It is not an absolute guarantee or a silver bullet, but it is definitely a big deal.

Herley and van Oorschot summarize similarly [25]:

The value of a formal guarantee is that it concentrates [remaining] doubt on the assumptions.

We conclude this perspective by noting that if proofs provide qualified guarantees they are not alone in doing so. Yet few other security assurance technologies are talked about in this same language. Consider fuzzing with the use of a memory corruption detector like LLVM's AddressSanitizer [39] as a means to discover memory corruption vulnerabilities and the desire for assurance that a fuzzed piece of software will exhibit no such vulnerabilities when deployed. Fuzzing provides this assurance qualified on the assumption that the software in deployment will not reach a state that it did not visit during fuzzing. For any non-trivial piece of software, the probability this assumption will hold is vanishingly small (based on intuition alone). Yet deciding whether the assumptions that underpin a particular proof are likely to hold in reality seems to be an entirely different problem.

When considering whether a proof provides any real-world guarantees, one *must* answer the question of whether its assumptions will hold in reality. We conjecture that the reason proofs are talked about in terms of providing unqualified guarantees is *not* because people find it easy to answer "yes" to this question. Rather, we suspect that those most likely to talk in this way are those least likely to attempt to ask this question at all.

1A) Proofs as Probabilistic Guarantees: A subcase of proofs as guarantees qualified by assumptions, is proofs providing guarantees (that some system property will hold in the real world) *quantified* by heuristically estimated likelihoods that the proof assumptions match reality. This idea is closely related to the use of formal methods for judging the *probability of perfection* of a piece of software [40]. Perfection captures the idea that some critical piece of software will *never* fail. Unlike correctness, perfection is not judged against a set of requirements or properties. Instead, "perfection includes a judgement that the requirements are the *right* requirements" [40] (emphasis in original).

In the context of proofs and security, perfection requires that the formal model and assumptions not only match the system's real-world behaviour always but also that of potential attackers. Perfection leaves no room for caveats like "up to modelling assumptions" and so on. Despite recent work [41] on methods for accurately estimating relevant probabilities, it remains unclear if this approach is viable. Quantification, absent large amounts of empirical data against which to validate quantitative assumptions and models, is necessarily fraught. As has been noted elsewhere, "little evidence supports the hypothesis that 'security can correctly be represented with quantitative information' [42]."

2) Proofs as Structured Exploration: An alternative perspective is that the value in proofs is not to provide (qualified) guarantees, but instead to force careful and rigorous understanding of a system (i.e., system model) and its operation. By doing so one might hope to find vulnerabilities during the process of performing a proof, or to better understand what actions might need to be taken to help ensure a system meets a particular security objective, and so on.

Shapiro provides one summation of this viewpoint [43]:

> There is no question in my mind that proof processes generate more robust code. Yet the general [consensus] seems to be that this robustness is much more an emergent consequence of rigorously understanding the problem [than] a result of the proof discharge.

If discharging the entire proof is merely a side effect, we note that under this point of view even *partial* proofs have value. The same is less clear for the viewpoint of proofs as qualified guarantees, since an unfinished proof provides no guarantees about the formal objects under study.

Interactive proofs—whether carried out in a proof assistant like HOL4 [37], Isabelle [44], Coq [45] or specialist security proof assistants like CryptoVerif [46] and EasyCrypt [47], or carried out using pen and paper (e.g. as in early authentication logics like BAN [48], or more modern proofs [49])—provide a critical mechanism for performing structured, interactive exploration of a system (via a formal model). Carrying out the proof forces the human to carefully examine each part of the system model, and allows human intuition to bring to light issues that might be missed by the proof itself (or to bring them to light before the proof has dead-ended with an unprovable subgoal). This is certainly one benefit of carrying out this style of proof. To some degree that same benefit is realised even in the context of fully automatic proofs (e.g., model checking [50] and automatic protocol analysers like ProVerif [51]), as even the process of building the formal model can uncover flaws [52]. (Note that automatic analysers also produce useful counter-examples.)

Yet one can go further, using the proof itself as a way to perform targeted exploration along a particular dimension of the model. We again use an example to illustrate.

Example 3: In November 2012, a 3.5 person-year effort to prove an information flow security theorem for the seL4 microkernel was completed [31]. This theorem was designed to provide evidence that seL4 could isolate mutually distrusting *partitions* and prevent unwanted information flows between them. As with the above-mentioned proofs of Khakpour [34], it did not reason about timing channels. When summarising the kinds of storage channels covered by the proof, Murray et al. were careful to state the limits of the theorem [31]:

> Our proof does not rule out the possibility of covert storage channels that are below the level of abstraction of [seL4's] abstract specification, but that the kernel never reads. For instance, suppose the kernel were ported to a new platform that included extra CPU registers that the kernel never reads, but that the port was done incorrectly such that the kernel fails to clear these registers on a partition switch. It is possible our proof would still hold despite the presence of an obvious covert storage channel.

Within a month of the text above being written, Anna Lyons discovered that seL4 had such a channel [53], present not on a port of seL4 to a new platform, but on the ARMv6 platform for which seL4 had been verified at that time. Specifically, the kernel failed to clear the load-exclusive monitor state (part of the aforementioned exclusive monitor state) on context switch. By using the `LDREX` and `STREX` instructions, one thread could conceivably signal to another, with whom the kernel was supposed to be preventing it from communicating. On its own, this example highlights the care with which proofs—as evidence about complex systems—must be treated.

The issue here was simply that the model of the ARM hardware as visible to the kernel was incomplete: it lacked this crucial piece of state since the kernel never made use of it. Within months work would begin on formally proving whether there existed other similar pieces of state that the kernel needed to clear on context switch [53], by reasoning over the validated ARM instruction set architecture model of Fox and Myreen [22]. That work formally stated and proved data integrity and confidentiality properties that, although somewhat more general, were very similar to those that Khakpour et al. [34] were already independently working on.

In the case of the seL4 proofs over Fox and Myreen's ISA model (which, while completed and subsequently formally connected to the kernel model [53], remain unpublished), the initial goal was simply to use proof as a way to rigorously explore the ARM ISA state as present in the ISA model. While a manual audit of the model might well have identified any extra user-visible state, formally stating and proving a data confidentiality property increased confidence none was missed. *(End of Example 3)*

3) Proofs as Commercially-valuable Commodities: Briefly, proofs and formal verification also have economic value.

Sales and marketing of commercial products. Formal verification can provide product differentiation. Some government departments are restricted by policy to endorsed or validated products. Non-experts interpret language like "proof" and "formally verified" to mean a product is safe from *all* attacks.

Liability cover. A verified product may be selected to limit commercial liability, by claim of compliance with standards or best practices (proof as best practice), whether or not there is belief in the guarantees of formal methods. The idea: no one loses their job for buying a formally verified product.

Leverage. Verification bolsters claims of superiority. Favoured proposals may be advanced over alternatives in standards based on claimed superiority backed by a proof. The assumption might be that proofs guarantee improved security in general, while in reality typically delivering targeted security properties under specific assumptions.

IT products evaluated under Common Criteria [54] level EAL7 require formally verified design and testing; a formal model is required for design assurance in crypto-module verification under FIPS 140-2 [55] (cf. ISO-19790 [56]).

III. Benefits, Drawbacks and Side Effects

A. General Benefits and Drawbacks

Our discussion of various perspectives on the role of proofs for security highlights some established ideas on their benefits: providing qualified guarantees, allowing auditors to concentrate effort on validating proof assumptions and formal model; providing a means for structured exploration of a system to better understand and improve its security; and potentially, allowing one to quantify the strength of security guarantees.

Proofs have helped in discovering security vulnerabilities and shining light on security issues that otherwise might have gone undetected. Lowe's attack on the Needham-Schroeder public key protocol [52] is a well-known example of formal methods to uncover and help fix a decades-old security (design) flaw. Other examples abound and we do not question proof's ability to improve security in general.

Yet against these benefits stand proof's disadvantages, which as with proof itself, remain poorly understood especially outside the formal methods community. One is proof *brittleness*: e.g., changing one line of code can potentially invalidate large amounts of formal reasoning, and also it is difficult to judge the guarantees provided by a proof when even *one* proof assumption deviates from reality (see Section IV). Another is a dearth of techniques for reliably predicting the cost of formal methods [57]. Other disadvantages mentioned earlier are the difficulty that both non-experts and experts have in discerning precisely the nature of the formal property that a proof establishes, and the difficulty of validating implicit assumptions in large-scale formal models on which such proofs rest. This leaves plenty of room for gaps between what non-experts might *think* has been proven and the precise formal property *actually* proven. For a timely concrete example, Cohn-Gordon and Cremers [58] consider recent work on formally verified secure messenger applications and the gap that exists "between the security properties that users might expect from a communication app, and the security properties formally proven" (cf. [59]).

B. Side Effects on the System

Most of these general pros and cons will be familiar to readers who have studied formal methods in the context of security. For the rest of this section, however, we turn attention to the effects *on the system*, both beneficial and otherwise, for which proof is a proximate cause—i.e., concrete effects on the system brought about directly as a result of performing the proof. We refer to these as the *side effects* of a proof.

A proof might *induce* two kinds of side effects on the system: 1) *changes to the code* of the system being proved secure, as a consequence—either to fix a vulnerability found during the proof, or to modify some aspect of design or implementation to open a path enabling a proof; 2) *deployment constraints* imposed on the system to enforce *environmental assumptions* necessary for proofs, related to controllable configuration and deployment aspects. For example, seL4's information flow proofs [31] require that seL4's interrupt delivery mechanism be disabled (forcing device drivers to poll for device interrupts). We separate environmental assumptions (subject to system control) from attack-model *expectations* (beyond system control), i.e., assumptions related to adversary capabilities; this also helps track the moral hazard of making proofs work simply by reducing attacker capabilities. A third category of assumptions we identify is *domain hypotheses*—these are not system-controllable, but assumed domain properties—e.g., that hardware performs to its specification (row-hammer attacks demonstrate this particular hypothesis is often false).

C. Relationships between Side Effects and the Real World

Here we discuss, with the aid of a set of Venn diagrams, possible relationships between these kinds of side effects on the system induced by the proof, on the one hand, and *those that improve the system's security by stopping real-world attacks*, on the other. At first glance one might reasonably expect the set of effects on the system induced by the proof to *necessarily* improve the system's resilience to real-world attacks. As we will argue, this position is far from clear.

For simplicity of exposition, we use the term *changes* to refer to both these kinds of side effects, i.e., code changes and changed deployment constraints. We consider changes made to software to enable proof of some property of the software, and whether those changes actually stop real attacks or simply facilitate the proofs (or both). In general, there will also be attacks that are possible but beyond the scope of a given formal proof; code changes that could stop such attacks may be beneficial to the real system, but are not needed for the formal verification in question. As an example, changes to close timing channels would be invisible to proofs of storage channel freedom.

What follows is a set of thought experiments presented to raise questions about the role of proof for building secure

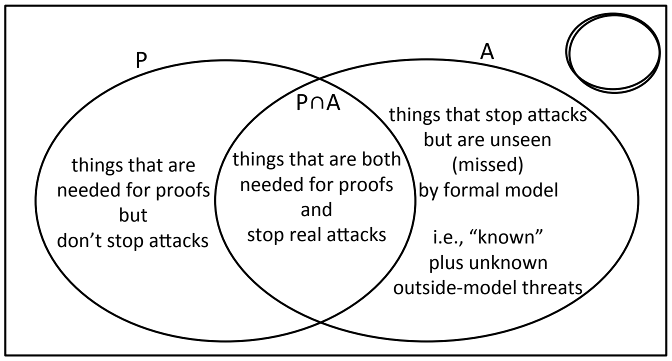
P
A
P∩A
things that are needed for proofs but don't stop attacks
things that are both needed for proofs and stop real attacks
things that stop attacks but are unseen (missed) by formal model
i.e., "known" plus unknown outside-model threats

Fig. 2. Scenario 1. P is the set of side effects needed to get proofs to work. A is the set of side effects needed to stop real-world attacks* (*relative to a specific target environment). Ideally, P = A with the rings coincident or nearly so (top right corner), conjectured as the mental model of most researchers.

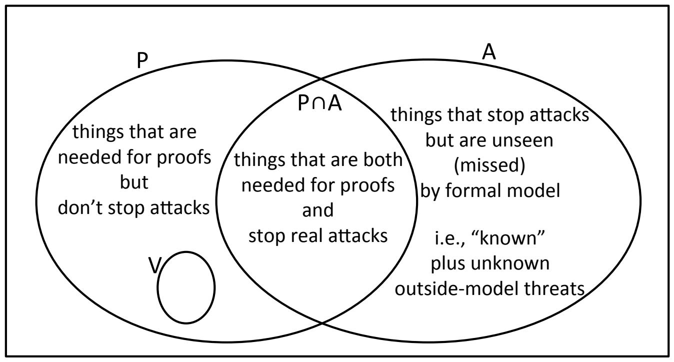
P
A
P∩A
things that are needed for proofs but don't stop attacks
V
things that are both needed for proofs and stop real attacks
things that stop attacks but are unseen (missed) by formal model
i.e., "known" plus unknown outside-model threats

Fig. 3. Scenario 2. This brings into discussion a set V (ideally empty): the subset of P that, contrary to intention, introduces vulnerabilities not seen by the proofs. For simplicity, V is shown disjoint from A, but it need not be—e.g., fixing a buffer overflow vulnerability might introduce a timing channel.

systems—questions having largely escaped previous scrutiny. As a first question, consider in relation to Figure 2 and some system and security proof for it: What is the expected size of the intersection between the set A of changes that stop real-world attacks, and the set P of changes that enable the proof?

The area of P that doesn't intersect A accounts for changes made for the sake of the proof that do not stop real-world attacks. These changes are side effects of the proof that don't actually lead to a real-world improvement of security. Instead they might impose performance penalties, or restrict functionality. If the number of such changes required for a proof is large, one might reasonably question the additional cost imposed on the system *by the proof process itself*.

We note that, in general, understanding a proof's side effects is particularly important if one or more of a proof's assumptions do not match reality. The proof then might provide a false sense of security (although, as Example 2 above demonstrates, this can be difficult to judge). Note that any changes made (and related performance costs incurred) to facilitate proofs are forced upon the user population whether or not the assumptions necessary for the proof to have value are met; when these are not met, the proof result is not delivered, but any costs incurred remain.

Figure 3 refines Figure 2, introducing the set V of *vulnerability inducing side effects*: the subset of P comprising all changes needed for the proofs that, contrary to intention, *introduce* vulnerabilities not seen by the proofs. As an example, Murray et al. [31] note that seL4's *partition scheduler* [31], introduced to enable a proof of storage channel freedom, might introduce certain timing channels.

Part (a) of Figure 4 considers a different possible relationship between P and A, with A a proper subset of P. The situation may be inefficient, but doesn't hurt security: real attacks are stopped, albeit extra "convenience" side effects (software artifacts and environmental assumptions introduced solely to enable proofs) may add unnecessary constraints, complexity or performance penalties to the system. We could summarize this as "defensively conservative, but safe".

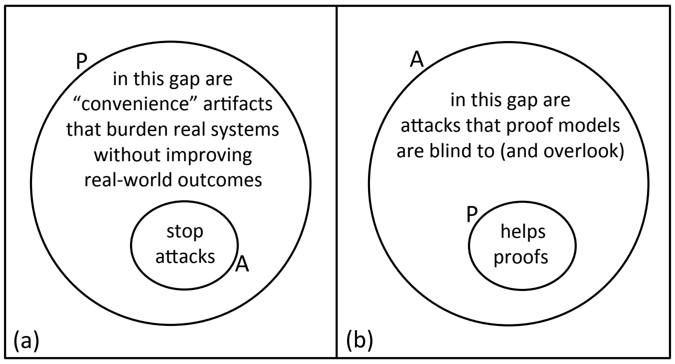
P
in this gap are "convenience" artifacts that burden real systems without improving real-world outcomes
stop attacks
A
(a)
A
in this gap are attacks that proof models are blind to (and overlook)
P
helps proofs
(b)

Fig. 4. Scenarios 3 and 4. Part (a) gives focus to software artifacts introduced to enable proofs, possibly at some cost (e.g., to software performance), but which do not stop any real-world attacks; as illustrated, A is a proper subset of P. The reverse case (b) has P a proper subset of A; see discussion inline.

Part (b) of Figure 4 inverts the relationship between P and A, showing the far more pessimistic case of a proof model blind to a great number of attacks (many are missed). As a small consolation, no software artifacts or environmental assumptions are added unnecessarily to facilitate proofs without providing relief from attacks. However here, the proofs largely provide a false sense of security, in that many attacks remain.

Figure 5 considers yet another theoretical possibility: that P and A are disjoint. This is a worst case: the environmental assumptions imposed and the software changes made to enable the proofs end up stopping *no* real attacks at all, while assumptions and changes needed to stop the real attacks are missed by the formal model. Here the proof has no positive impact on security, producing side effects without any benefits.

Our motivation is to raise awareness of possibilities. Naturally, which of these scenarios is more likely will be different for each verified system, and will depend on the nature of the system, its formal model, and the properties established by the proofs. Some of these scenarios may be rare or non-existent in practice, but we note: there is little, if any, detailed discussion of these possibilities in the literature. A question

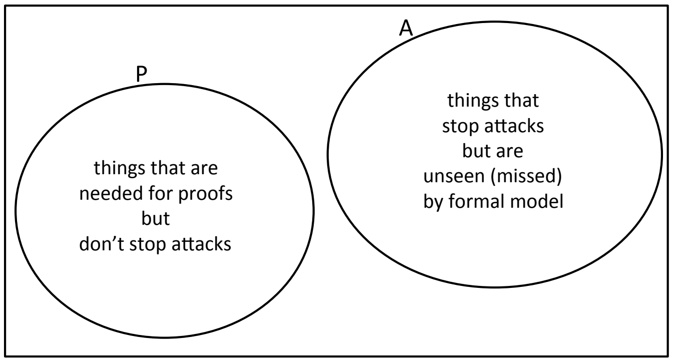

Fig. 5. Scenario 5. This considers a very pessimistic case of P and A having no intersection. For a definition of P and A, see caption of Figure 2.

we return to is: do we have *any* idea about (how to determine) the relationships between P and A in Figure 2 or any others in this series of Venn diagrams, or any sense of the sizes of these sets and their intersection?

IV. Discussion of Side Effects

As noted, a common form of side effect of formal verification is *deployment constraints* imposed on a system, e.g., to reflect environmental assumptions encoded in the formal proofs or the model over which they are carried out. Often these restrictions are sensible and arguably enhance security in many scenarios. Yet for other reasonable threat models the restrictions might be overkill. We use an example to illustrate.

Example 4: As mentioned in Section III, the seL4 information flow proofs explicitly assume that seL4's mechanism for delivering device interrupts to application programs is disabled. This is a sensible restriction if one cares about strong isolation since as noted [31], "the [seL4] kernel does not isolate the interrupts of one partition from another".

Yet in many scenarios one might reasonably care about isolating memory without worrying about information leakage via interrupts, e.g., in systems where all application components are written by trusted authors, and verified but only to detect *unintentional* information leakage. The Cross Domain Desktop Compositor is an example [60], [61].

Here one might hope to use seL4's information flow theorem to conclude that the seL4 kernel does indeed adequately isolate memory, if not interrupts. Yet, as the top level theorem is stated, it provides no such assurance since the theorem explicitly assumes that all device interrupts are disabled.

One might reasonably expect that that theorem has a variant "hiding inside it" that would provide the desired assurance; yet no such theorem has yet been proved and it isn't clear what level of effort would be required to prove it (although one might expect to be able to re-use lemmas used to establish the existing information flow theorem), if it is even provable. As it is, seL4's confidentiality guarantees as visible through its top level information flow theorem come at the price of disabling device interrupts—a severe deployment restriction for many. *(End of Example 4)*

A similar issue arises when a system or protocol is modified to prove it secure. We highlight with another example.

Example 5: Dowling and Paterson [49] recently presented a proof of the WireGuard protocol [62]. To allow the key exchange phase and the data transport phase of the protocol to be analysed separately using standard techniques, they modified the protocol, introducing an extra message transmission. This naturally raises questions: Was the original pre-modified protocol secure? What does their proof mean for it? Does the added message stop a real world attack or is it merely a change for proof convenience? Quoting their work [49]:

> WireGuard either cannot be proven secure as a key exchange protocol using standard key-indistinguishability notions, or it is vulnerable to key-recovery attacks in the [Key Compromise Impersonation] setting.

Which is it? This is unclear to outsiders and requires more than cursory examination by experts [63]. Moreover, the formal proofs themselves do not help answer this question. Relying on the proofs alone as evidence, one pays the penalty of an extra message transmission, defeating the one-round (1-RTT) feature of WireGuard's key exchange phase.
(End of Example 5)

We note a difficulty here in judging the value of a proof regarding a protocol intentionally deviating from the original, and which has no implemented counterpart in the real world—a *dangling proof* (as coined by Virgil Gligor). We might use this term also for proofs having assumption sets which are met in no deployed system. Another difficulty arises in taking proofs performed about software in one context and updating them to apply to the software in other, less restrictive contexts—for example, as found in exploring the application of Green Hills' certified INTEGRITY separation kernel to commodity platforms [64]. While modern interactive proof assistants help to manage the complexity of this task, as demonstrated, e.g., by the ongoing work to extend the verification of the seL4 kernel to cover various hardware platforms [65], it remains non-trivial, especially for large proofs about real-world software.

V. Related Work

Mention of related work is interspersed throughout this paper, including for security-related formal verification literature and in particular numerous references to the seL4 project. That project is perhaps rivalled in literature and practicality only by the growing work on formal methods specifically related to TLS (e.g., Beurdouche [66], Delignat-Lavaud [67], and for HTTPS, Bhargavan [68]). For a comprehensive summary of the state-of-the-art of formal verification related to TLS 1.3, we recommend Bhargavan [69] rather than repeating it here.

On the specific topic of the meaning and value of proofs, Asperti et al. [70] revisit the critique of formal verification by DeMillo et al. [2] thirty years later, and among other things note unanticipated contributions of interactive theorem provers; Regehr [18] gives links to enthusiastic discussion of this topic. Koblitz takes up controversial discussion of the meaning of proofs and differing views of mathematicians and theoretical cryptographers [19] (cf. [3]).

The role of formal methods features prominently in government efforts towards a Science of Security [71]; see Herley and van Oorschot [25] for background. Degabriele et al. [59] discuss the gap between crypto proofs and end-user expectations (cf. [58], above). In a theory of security testing, Torabi Dashti and Basin [26] distinguish system specifications (desired system behaviour) from security requirements (desired properties of the system's world), and explicate difficulties in reasoning about adversarial environments, e.g., the scope-limiting *closed-world assumption* inherent in models. Jackson's requirements engineering framework and reference model [72] relates software systems to their environments in the context of designing a *machine* deployed in a *world*; see van Lamsweerde [27] for perspective, and the role of domain hypotheses and assumptions in satisfaction arguments.

VI. Concluding Discussion

We conclude with some questions following directly from earlier sections. Our hope is that these stimulate interesting discussion and encourage others to give them thought.

As discussed, proofs concretely impact software systems with two types of side effects: code changes, and deployment constraints. While purists may value proofs *per se*, practitioners value changes that stop actual attacks. This leads to:

Q1: *Can we find means to know and measure the relationship between proof side effects and changes that stop attacks, how these sets intersect, and the intersection sizes?*

Not knowing this precludes weighing the benefits delivered by our formal proofs, against the concrete side effect costs imposed (beyond proof effort costs).

We have discussed various meanings and values of proofs, including proofs as guarantees predicated on assumptions—the guarantees conditional on proof assumptions carrying over to real systems. This fine print is evidently entirely inaccessible to non-experts, and even for experts, formal theorems are labour-intensive to understand and assumptions are often buried deep within formal models. When one or more such adversary assumptions, domain hypotheses, or configuration/deployment constraints do not hold in a real system, the residual value of software verification proofs is little understood—does it entirely evaporate, if a single assumption from a long list fails? We expect otherwise in large, multiple person-year software verification efforts, where it appears naive to expect that every required proof assumption holds in the practical system. This leads to the question:

Q2: *Can we find means to measure the residual value of proofs, when not all assumptions hold in practice; can we presently even begin to attempt such a measurement?*

Finally, and sadly, we turn to practitioners and all non-experts, the unhappy recipients of the news that "proof" does not really mean 100% guarantee, in the sense expected. We ask, for this long-standing issue, which reliably misleads almost everyone:

Q3: *How can we better tag formally verified software to explain the fine print that accompanies the proofs?*

A closely-related question is:

Q4: *What effort can be undertaken to explore formal or other methods to track and validate that (both implicit and explicit) security assumptions in large-scale formal models hold in practice?*

Outside of certification labs, few are incentivised to track and validate these assumptions and there appears to be a void in terms of both culture and process for doing so.

Without answers to such questions, it is hard to say what empirical value our formal proofs deliver to real systems. That precludes providing convincing cost-benefit analyses. We believe that finding answers to some, or all, of these questions, can significantly advance the cause of formal verification of software in particular, and formal methods in general.

Acknowledgements

We thank David Basin, Virgil Gligor, Cormac Herley, Benjamin Pierce and John Regehr for insightful feedback, and anonymous referees. The second author is Canada Research Chair in Authentication and Computer Security, and acknowledges NSERC for funding the chair and a Discovery Grant.

References

[1] D. MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust*. Cambridge, MA, USA: MIT Press, 2001.

[2] R. A. De Millo, R. J. Lipton, and A. J. Perlis, "Social processes and proofs of theorems and programs," *Commun. ACM*, vol. 22, no. 5, pp. 271–280, 1979, also letters to the editor: C.ACM vol.22 no.11, 621–630.

[3] N. Koblitz and A. Menezes, "Another look at "provable security"," *J. Cryptology*, vol. 20, pp. 3–37, 2007.

[4] M. Y. Vardi, "The automated-reasoning revolution: from theory to practice and back," Distinguished Lecture at NSF CISE, Spring 2016.

[5] G. Heiser, T. Murray, and G. Klein, "It's time for trustworthy systems," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 67–70, 2012.

[6] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," in *ACM SOSP*, 2009, pp. 207–220.

[7] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *IEEE EuroS&P*, 2017, pp. 435–450.

[8] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub, "Implementing TLS with verified cryptographic security," in *IEEE Symp. Security and Privacy*, 2013, pp. 445–459.

[9] D. Jang, Z. Tatlock, and S. Lerner, "Establishing browser security guarantees through formal shim verification," in *USENIX Security*, 2012.

[10] S. Kanav, P. Lammich, and A. Popescu, "A conference management system with verified document confidentiality," in *International Conference on Computer Aided Verification*. Springer, 2014, pp. 167–183.

[11] T. Bauereiß, A. P. Gritti, A. Popescu, and F. Raimondi, "CoSMed: A confidentiality-verified social media platform," in *International Conference on Interactive Theorem Proving*. Springer, 2016, pp. 87–106.

[12] A. Azevedo de Amorim, N. Collins, A. DeHon, D. Demange, C. Hriţcu, D. Pichardie, B. C. Pierce, R. Pollack, and A. Tolmach, "A verified information-flow architecture," *J. Computer Security*, vol. 24, no. 6, pp. 667–688, 2016.

[13] R. Gu, Z. Shao, H. Chen, X. N. Wu, J. Kim, V. Sjöberg, and D. Costanzo, "CertiKOS: An extensible architecture for building certified concurrent OS kernels," in *OSDI*, Nov. 2016.

[14] C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang, and B. Zill, "Ironclad apps: End-to-end security via automated full-system verification," in *OSDI*, Oct. 2014, pp. 165–181.

[15] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. T. V. Setty, and B. Zill, "IronFleet: proving practical distributed systems correct," in *ACM SOSP*, 2015, pp. 1–17.

[16] T. Bauereiß, A. P. Gritti, A. Popescu, and F. Raimondi, "CoSMeDis: a distributed social media platform with formally verified confidentiality guarantees," in *IEEE Symp. Security and Privacy*, 2017, pp. 729–748.

[17] K. Q. Ye, M. Green, N. Sanguansin, L. Beringer, A. Petcher, and A. W. Appel, "Verified correctness and security of mbedTLS HMAC-DRBG," in *ACM CCS*, 2017, pp. 2007–2020.

[18] J. Regehr, "Research for practice: vigorous public debates in academic computer science," *Commun. ACM*, vol. 60, no. 12, pp. 48–50, 2017.

[19] N. Koblitz, "The uneasy relationship between mathematics and cryptography," *Notices of the AMS*, vol. 54, no. 8, pp. 972–979, 2007.

[20] T. Sewell, M. Myreen, and G. Klein, "Translation validation for a verified OS kernel," in *PLDI*, 2013, pp. 471–481.

[21] R. Kumar, M. O. Myreen, M. Norrish, and S. Owens, "CakeML: a verified implementation of ML," in *ACM Principles of Programming Languages (POPL)*, 2014, pp. 179–191.

[22] A. Fox and M. Myreen, "A trustworthy monadic formalization of the ARMv7 instruction set architecture," in *Interactive Theorem Proving (ITP)*, ser. Springer LNCS, vol. 6172, 2010, pp. 243–258.

[23] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, 2014, pp. 361–372.

[24] S. Chong, J. Guttman, A. Datta, A. Myers, B. Pierce, P. Schaumont, T. Sherwood, and N. Zeldovich, "Report on the NSF Workshop on Formal Methods for Security," 2016.

[25] C. Herley and P. C. van Oorschot, "SoK: Science, security and the elusive goal of security as a scientific pursuit," in *IEEE Symp. Security and Privacy*. IEEE, 2017, pp. 99–120.

[26] M. Torabi Dashti and D. A. Basin, "Security testing beyond functional tests," in *Engineering Secure Soft. and Systems (ESSoS)*, 2016, pp. 1–19.

[27] A. van Lamsweerde, "From Worlds to Machines (A Tribute to Michael Jackson)," 2009, pp. 1–13, manuscript.

[28] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," meltdownattack.com, 2018.

[29] "seL4 frequently asked questions: The proof," https://sel4.systems/Info/FAQ/proof.pml, 2018.

[30] N. Koblitz and A. Menezes, "Another look at "provable security". II," in *International Conference on Cryptology in India*, 2006, p. 148175.

[31] T. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein, "seL4: from general purpose to a proof of information flow enforcement," in *IEEE Symp. Security and Privacy*, 2013, pp. 415–429.

[32] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser, "Comprehensive formal verification of an OS microkernel," *ACM TOCS*, vol. 32, no. 1, pp. 2:1–2:70, Feb 2014.

[33] Benjamin C. Pierce, personal email correspondence, 25 Jun 2018, reported with permission.

[34] N. Khakpour, O. Schwarz, and M. Dam, "Machine assisted proof of ARMv7 instruction level isolation properties," in *International Conference on Certified Programs and Proofs*, 2013, pp. 276–291.

[35] G. Klein, T. Murray, P. Gammie, T. Sewell, and S. Winwood, "Provable security: How feasible is it?" in *USENIX HotOS workshop*, May 2011.

[36] *ARM Architecture Reference Manual, ARM v7-A and ARM v7-R*, ARM Ltd., Apr. 2008, ARM DDI 0406B.

[37] "HOL4," http://hol.sourceforge.net.

[38] G. Klein, "Correct OS kernel? Proof? Done," *USENIX login*, vol. 34, no. 6, pp. 28–34, 2009.

[39] K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "AddressSanitizer: A fast address sanity checker." in *USENIX Annual Technical Conf.*, 2012, pp. 309–318.

[40] J. Rushby, "Software verification and system assurance (invited paper)," in *IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, 2009, pp. 3–10.

[41] X. Zhao, "On the probability of perfection of software-based systems," Ph.D. dissertation, City University of London, 2016.

[42] V. Verendel, "Quantified security is a weak hypothesis: a critical survey of results and assumptions," in *New Security Paradigms Workshop (NSPW)*, 2009, pp. 37–50.

[43] J. S. Shapiro, "On the (alleged) value of proof for assurance," *Lambda The Ultimate*, posted. http://lambda-the-ultimate.org/node/3858, 2010.

[44] T. Nipkow, L. Paulson, and M. Wenzel, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, ser. Springer LNCS, 2002, vol. 2283.

[45] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, ser. Texts in Theoretical Computer Science. An EATCS Series, 2004.

[46] B. Blanchet, "A computationally sound mechanized prover for security protocols," *IEEE TDSC*, vol. 5, no. 4, pp. 193–207, 2008.

[47] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, "EasyCrypt: A tutorial," in *Foundations of Security Analysis and Design VII*. Springer, 2014, pp. 146–166.

[48] M. Burrows, M. Abadi, and R. M. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, pp. 18–36, 1990.

[49] B. Dowling and K. G. Paterson, "A cryptographic analysis of the WireGuard protocol," 2018, https://eprint.iacr.org/2018/080.

[50] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT press, 1999.

[51] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[52] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1996, pp. 147–166.

[53] T. Murray, "The quest for assured confidentiality: Keeping secrets with seL4," Univ. Oxford talk. Video: https://www.cybersecurity.ox.ac.uk/resources/videos/the-quest-for-assured-confidentiality, Jan. 2015.

[54] ISO/IEC, "ISO/IEC 15408: Common Criteria for Information Technology Security Evaluation," Apr 2017, version 3.1, revision 5 ("CC 3.1"). Part 1: Introduction and general model. Part 2: Security functional components. Part 3: Security assurance components.

[55] NIST, "FIPS Pub 140-2: Security Requirements for Cryptographic Modules," U.S. Dept. of Commerce, Federal Information Processing Standards Publication, May 25, 2001.

[56] ISO/IEC, "ISO/IEC 19790: Information Technology - Security Techniques - Security requirements for cryptographic modules," Aug 2012, corrected Nov 2015.

[57] D. Matichuk, T. Murray, J. Andronick, R. Jeffery, G. Klein, and M. Staples, "Empirical study towards a leading indicator for cost of formal software verification," in *ICSE*, 2015, pp. 722–732.

[58] K. Cohn-Gordon and C. Cremers, "Mind the gap: Where provable security and real-world messaging don't quite meet," Oct. 2017. [Online]. Available: https://eprint.iacr.org/2017/982

[59] J. Degabriele, K. Paterson, and G. Watson, "Provable security in the real world," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 33–41, 2011.

[60] T. Murray, R. Sison, and K. Engelhardt, "COVERN: A logic for compositional verification of information flow control," in *IEEE EuroS&P*, 2018, to appear.

[61] M. Beaumont, J. McCarthy, and T. Murray, "The Cross Domain Desktop Compositor: Using hardware-based video compositing for a multi-level secure user interface," in *ACSAC*, 2016, pp. 533–545.

[62] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel," in *NDSS*, 2017.

[63] M. D. Green, "Correspondence on Twitter," Jan. 2018, https://twitter.com/matthew_d_green/status/956151576304996352.

[64] NSA, Information Assurance Directorate, Systems and Network Analysis Center, "Separation kernels on commodity workstations," Mar. 2010.

[65] "seL4Wiki: Supported hardware platforms," https://wiki.sel4.systems/Hardware, Dec. 2017.

[66] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue, "A messy state of the union: taming the composite state machines of TLS," *Commun. ACM*, vol. 60, no. 2, pp. 99–107, 2017.

[67] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Z. Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue, "Implementing and proving the TLS 1.3 record layer," in *IEEE Symp. Security and Privacy*, 2017, pp. 463–482.

[68] K. Bhargavan et al., "Everest: Towards a verified, drop-in replacement of HTTPS," in *Summit on Adv. in Prog. Lang., SNAPL*, 2017, pp. 1:1–1:12.

[69] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *IEEE Symp. Security and Privacy*, 2017, pp. 483–502.

[70] A. Asperti, H. Geuvers, and R. Natarajan, "Social processes, program verification and all that," *Math. Struct. C.S.*, 2009, 19(5):877-896.

[71] U.S. National Academies of Sciences, Eng., and Medicine, "Foundational Cybersecurity Research: Improving Science, Engineering, and Institutions," 2017, Nat. Academies Press, https://doi.org/10.17226/24676.

[72] M. Jackson, "The World and the Machine," in *ICSE*, 1995, pp. 1–10.