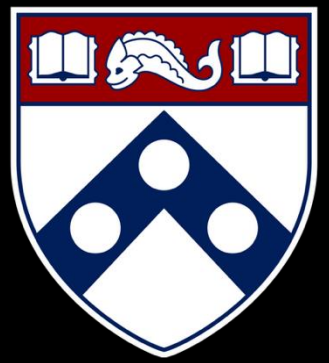
Secure Systems Engineering and Management

A Data-driven Approach



A Science of Security

Michael Hicks

UPenn CIS 7000-003
Spring 2026

(optional)

SoK: Science, Security, and the Elusive Goal of Security as a Scientific Pursuit

Cormac Herley
Microsoft Research, Redmond, WA, USA
cormac@microsoft.com

P.C. van Oorschot
Carleton University, Ottawa, ON, Canada
paulv@scs.carleton.ca

***Abstract*—The past ten years has seen increasing calls to make security research more "scientific". On the surface, most agree that this is desirable, given universal recognition of "science" as a positive force. However, we find that there is little clarity on what "scientific" means in the context of computer security research, or consensus on what a "Science of Security" should look like. We selectively review work in the history and philosophy of science and more recent work under the label "Science of Security". We explore what has been done under the theme of relating science and security, put this in context with historical science, and offer observations and insights we hope may motivate further exploration and guidance. Among our findings are that practices on which the rest of science has reached consensus appear little used or recognized in security, and a pattern of methodological errors continues unaddressed.**

***Index Terms*—security research; science of security; history of science; philosophy of science; connections between research and observable world.**

I. Introduction and Overview

Security is often said to have unique challenges. Progress can be harder to measure than in areas where, e.g., performance metrics or capabilities point to visible steady improvement. Supposedly unique factors, such as the presence of active adversaries, complicate matters. Some even describe the field in pessimistic terms. Multics warriors remind the young that many of today's problems were much better addressed forty years ago [1]. Shamir, in accepting the 2002 Turing award, described non-crypto security as "a mess." Schell, in 2001, described the field as being filled with "pseudo-science and flying pigs" [2].

Perhaps in response to these negative views, over the last decade there has been an effort in parts of the community to develop a "Science of Security" (SoS). In this paper we review both work in the history/philosophy of science and, recently, under this SoS banner. We wish to distinguish at the outset between these two strands. The first is an exploration of the techniques that the consensus from other fields suggest are important to pursuing any problem scientifically. The second is the activity and body of work that has resulted from external promotion of an agenda by the name "Science of Security", research) in the light of consensus views of science and scientific methods. We find that aspects from the philosophy of science on which most other communities have reached consensus appear surprisingly little used in security, including in work done under the SoS label. For example, we do not find that that work better adheres to scientific principles than other security research in any readily identifiable way.

We identify several opportunities that may help drive security research forward in a more scientific fashion, and on this we are cautiously optimistic. While we see great benefit to this, we also do not wish to argue that all of security must be done on rigidly scientific principles. A significant component of security is engineering; this shares with science the regular contact with, and feedback from, observation, despite not having as clearly articulated a definition or methods.

Section II selectively reviews literature on the history and philosophy of science, with particular emphasis on three things: 1) methodologies and positions on which practicing scientists and philosophers of science have largely reached consensus; 2) aspects highlighting opportunities to eliminate confusion in security research; and 3) contributions pointing to where security research might be made "more scientific". Section III selectively reviews literature relating "science" and "security", for examples of viewpoints within the community, for context in later discussion, and as supporting evidence for arguments; an exhaustive review of all security literature attempting to determine which papers use scientific methods in security research is not a goal. Section IV highlights areas where the security community has failed to adopt accepted lessons from the science literature. Section V provides insights and offers observations and constructive suggestions. Section VI concludes.

II. History/Philosophy of Science

This section highlights aspects from the history and philosophy of science most relevant to security research. Our goal here is not an encyclopedic review of science literature;

Blueprint for a science of cybersecurity

Fred B. Schneider

1. Introduction

A secure system must defend against all possible attacks—including those unknown to the defender. But defenders, having limited resources, typically develop defenses only for attacks they know about. New kinds of attacks are then likely to succeed. So our growing dependence on networked computing systems puts at risk individuals, commercial enterprises, the public sector, and our military.

The obvious alternative is to build systems whose security follows from first principles. Unfortunately, we know little about those principles. We need a *science of cybersecurity* (see box 1) that puts the construction of secure systems onto a firm foundation by giving developers a body of laws for predicting the consequences of design and implementation choices. The laws should

- transcend specific technologies and attacks, yet still be applicable in real settings,
- introduce new models and abstractions, thereby bringing pedagogical value besides predictive power, and
- facilitate discovery of new defenses as well as describe non-obvious connections between attacks

vulnerabilities in deployed systems and beyond the development of defenses for specific attacks. Yet, use of a science of cybersecurity when implementing a system should not be equated with implementing absolute security or even with concluding that security requires perfection in design and implementation. Rather, a science of cybersecurity would provide—independent of specific systems—a principled account for techniques that work, including assumptions they require and ways one set of assumptions can be transformed or discharged by another. It would articulate and organize a set of abstractions, principles, and trade-offs for building secure systems, given the realities of the threats and of our cybersecurity needs.

BOX 1. What is a science?

The term *science* has evolved in meaning since Aristotle used it to describe a body of knowledge. To many, it connotes knowledge obtained by systematic experimentation, so they take that process as the defining characteristic of a science. The natural sciences satisfy this definition.

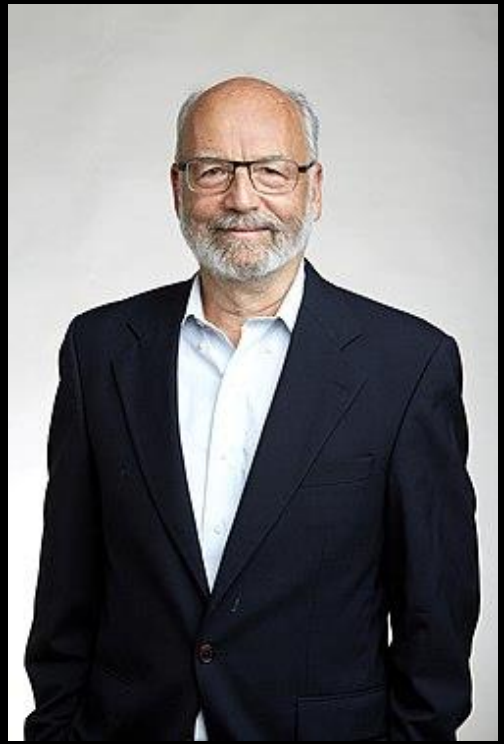
Experimentation helps in forming and then affirming theories or laws that are intended to offer verifiable predictions about man-made and natural phenomena. It is but a small step from science as experimentation to science as laws that accurately predict phenomena. The status of the natural sciences

Can we escape the security mess?

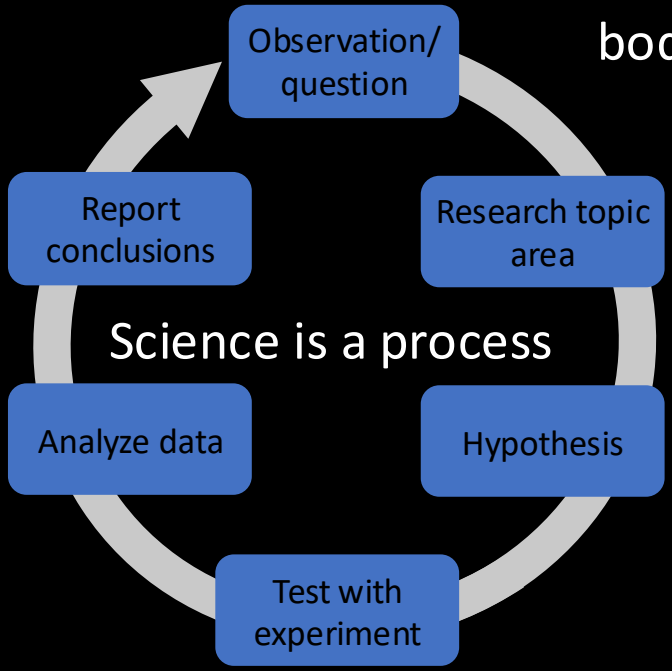
“In 10 years non-crypto security will remain a mess.”
— Adi Shamir (co-inventor of RSA), in 2002

How to improve this state of affairs?

Maybe: Seek to develop cybersecurity **science**



What is science?



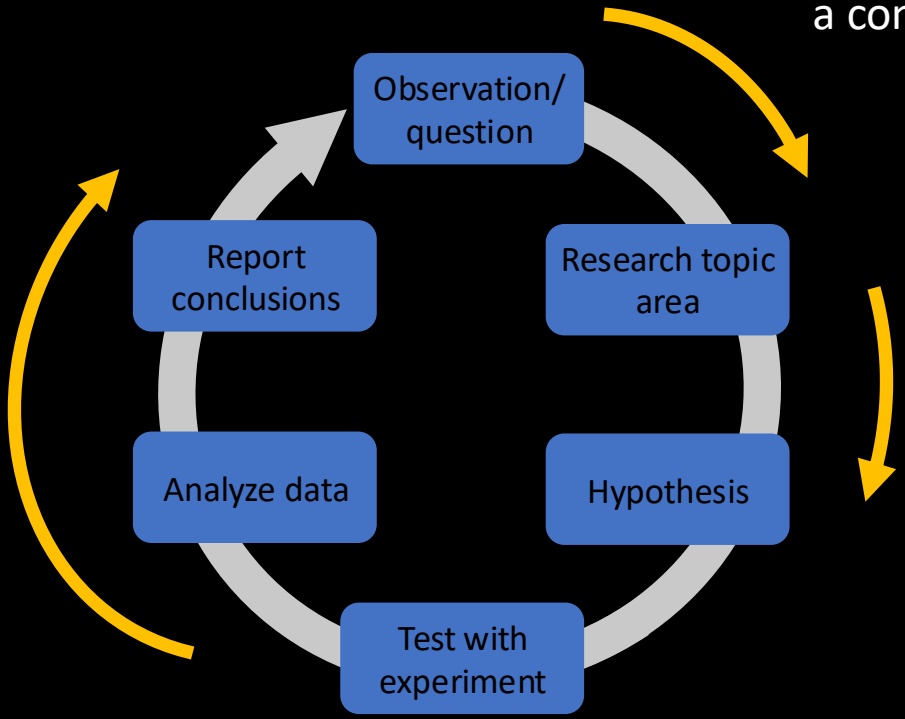
Science is a body of knowledge

The opening question

“In what ways is cybersecurity (not) a science?”

What is the scientific process?

Induction: generalizing from observation, constructing or refining a consistent model

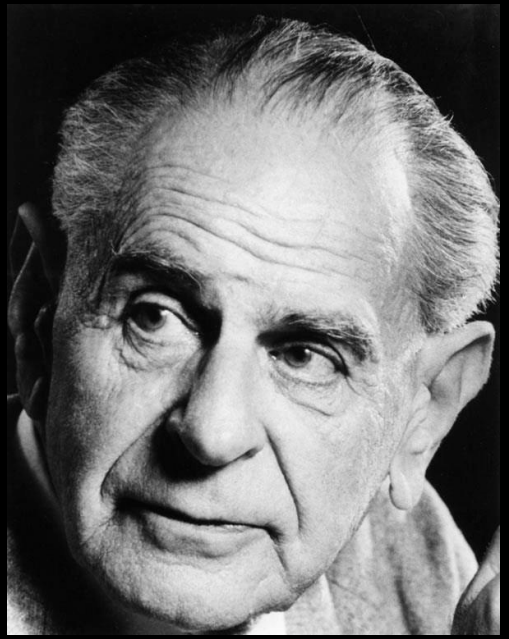


Falsification: Confirming or refuting a prediction against reality

Deduction: making a prediction from the model

Karl Popper's view: Falsifiability is critical

"A theory which is not refutable by any conceivable event is non-scientific. Irrefutability is not a virtue of a theory (as people often think) but a vice"



2008

Context: Silver bullets, not lemons

Extends Anderson:

- Security is a good
- Security is hard to assess
  - Leads to information insufficiency *on both sides*

Lacks empirical basis, i.e., *falsifiability*

The Market for Goods, as described by Information and by Party	**Buyer Knows**	**Buyer Lacks *H1***
Seller Knows	Efficient Goods	Lemons (used cars)
Seller Lacks *H2*	Limes (Insurance)	**Silver Bullets (Security)**

2012

NSA Science of Security Research Initiative

The Science of Security and Privacy (SoS) Initiative at the National Security Agency Research Directorate promotes foundational cybersecurity science that is needed to mature the cybersecurity discipline and to underpin advances in cyberdefense. Beginning in 2012, one part of the initiative is to fund foundational research at "Lablets." With emphasis on building a community, each lablet created partnerships with other universities called "Sub-Lablets." Science of Security researchers often freely collaborated with researchers in other institutions worldwide. The SURE project was founded to investigate cybersecurity in the cyber-physical systems realm and ran from 2014 to 2018. In 2018, CPS research was folded into the Lablet Research.

The SoS Initiative has defined 5 Hard Problems as the research areas for the initiative and community to work on. The problems are: 1) Resilience Architectures, 2) Scalability and Composability, 3) Metrics, 4) Secure Collaboration and 5) Human Behavior.

Annual Reports: Index | 2015 | 2016 | 2017 | 2018 | 2019 | 2021 | 2022

2013

Memory Trace Oblivious Program Execution

Chang Liu, Michael Hicks, and Elaine Shi
The University of Maryland, College Park, USA

***Abstract*—Cloud computing allows users to delegate data and computation to cloud service providers, at the cost of giving up physical control of their computing infrastructure. An attacker (e.g., insider) with physical access to the computing platform can perform various physical attacks, including probing memory buses and cold-boot style attacks. Previous work on secure (co-)processors provides hardware support for memory encryption and prevents direct leakage of sensitive data over the memory bus. However, an adversary snooping on the bus can still infer sensitive information from the memory access traces. Existing work on Oblivious RAM (ORAM) provides a solution for users to put all data in an ORAM; and accesses to an ORAM are obfuscated such that no information leaks through memory access traces. This method, however, incurs significant memory access overhead.**

This work is the first to leverage programming language techniques to offer efficient memory-trace oblivious program execution, while providing formal security guarantees. We formally define the notion of memory-trace obliviousness, and provide a type system for verifying that a program satisfies this property. We also describe a compiler that transforms a program into a structurally similar one that satisfies memory trace obliviousness. To achieve optimal efficiency, our compiler partitions variables into several small ORAM banks rather than one large one, without risking security. We use several example programs to demonstrate the efficiency gains our compiler achieves in comparison with the naive method of placing all variables in the same ORAM.

I. Introduction

Cloud computing allows users to delegate their data and computation to computing service providers, and thus relieves users from the necessity to purchase and maintain requisite computing infrastructure. The value proposition is appealing to both cloud providers and clients: market research predicts a 50% compound annual growth rate on public cloud workloads [23].

Despite its increasing popularity, privacy concerns have become a major barrier in furthering cloud adoption. Cloud customers offloading computations transfer both their code and their data to the provider, and thereby relinquish control over both their intellectual property and their private information. While various existing works have considered how to secure sensitive data in the cloud against remote software attacks [15, 31, 48], we consider a stronger adversarial

Previous work has proposed the idea of using memory encryption to ensure confidentiality of sensitive memory contents [33, 39, 40, 43, 44]. However, as memory addresses are transferred in cleartext over the memory buses, an adversary can gain sensitive information by observing the memory addresses accessed. For example, address disclosure can leak implicit program execution flows, resulting in the leakage of sensitive code or data [50].

Oblivious RAM (ORAM), first proposed by Goldreich and Ostrovsky [17], can be used to protect memory access patterns. In particular, we can place sensitive code and data into ORAM, and doing so has the effect of hiding the access pattern. Roughly speaking, this works by issuing many physical reads/writes for each logical one in the program, and by shuffling the mapping between the logical data and its actual physical location. Unfortunately, placing all code and sensitive data in ORAM leads to significant memory access overhead in practice [12, 42, 47], and can still leak information, e.g., according to the length of the memory trace. On the other hand, customized data-oblivious algorithms have been suggested for specific algorithms, to achieve asymptotically better overhead than generic ORAM simulation [11, 20]. This approach, however, does not scale in terms of human effort.

Contributions. We make four main contributions. First, we define *memory trace obliviousness*, a property that accounts for leaks via the memory access trace. We present a formal semantics for a simple programming language that allocates its secret data and instructions in ORAM, and define the trace of data reads/writes and instruction fetches during execution. We define memory trace obliviousness as an extension of *termination-sensitive noninterference* [2, 34] that accounts for the memory trace as a channel of information. Note that memory trace obliviousness is stronger than the notion used in the traditional ORAM literature [17]—it ensures the content and *length* of the memory access pattern are independent of sensitive inputs, while traditional ORAM security does not provide the latter guarantee.

Second, we present a novel type system for enforcing memory trace obliviousness, building on standard type sys-

2018

Evaluating Fuzz Testing

George Klees, Andrew Ruef, Benji Cooper
University of Maryland

Shiyi Wei
University of Texas at Dallas

Michael Hicks
University of Maryland

ABSTRACT

Fuzz testing has enjoyed great success at discovering security critical bugs in real software. Recently, researchers have devoted significant effort to devising new fuzzing techniques, strategies, and algorithms. Such new ideas are primarily evaluated experimentally so an important question is: What experimental setup is needed to produce trustworthy results? We surveyed the recent research literature and assessed the experimental evaluations carried out by 32 fuzzing papers. We found problems in every evaluation we considered. We then performed our own extensive experimental evaluation using an existing fuzzer. Our results showed that the general problems we found in existing experimental evaluations can indeed translate to actual wrong or misleading assessments. We conclude with some guidelines that we hope will help improve experimental evaluations of fuzz testing algorithms, making reported results more robust.

CCS CONCEPTS

• **Security and privacy → Software and application security**;

KEYWORDS

fuzzing, evaluation, security

ACM Reference Format:
George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. 2018. Evaluating Fuzz Testing. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3243734.3243804

1 INTRODUCTION

A *fuzz tester* (or *fuzzer*) is a tool that iteratively and randomly generates inputs with which it tests a target program. Despite appearing "naive" when compared to more sophisticated tools involving SMT solvers, symbolic execution, and static analysis, fuzzers are surprisingly effective. For example, the popular fuzzer AFL has been used to find hundreds of bugs in popular programs [1]. Comparing AFL head-to-head with the symbolic executor *angr*, AFL found 76% more bugs (68 vs. 16) in the same corpus over a 24-hour period [50]. The success of fuzzers has made them a popular topic of research.

Why do we think fuzzers work? While inspiration for new ideas may be drawn from mathematical analysis, fuzzers are primarily evaluated experimentally. When a researcher develops a new fuzzer algorithm (call it A), they must empirically demonstrate that it provides an advantage over the status quo. To do this, they must choose:

- a compelling *baseline* fuzzer B to compare against;
- a sample of target programs—the *benchmark suite*;
- a *performance metric* to measure when A and B are run on the benchmark suite; ideally, this is the number of (possibly exploitable) bugs identified by crashing inputs;
- a meaningful set of *configuration parameters*, e.g., the *seed file* (or files) to start fuzzing with, and the *timeout* (i.e., the duration) of a fuzzing run.

An evaluation should also account for the fundamentally random nature of fuzzing: Each fuzzing run on a target program may produce different results than the last due to the use of randomness. As such, an evaluation should measure *sufficiently many trials* to sample the overall distribution that represents the fuzzer's performance, using a *statistical test* [38] to determine that A's measured improvement over B is real, rather than due to chance.

Failure to perform one of these steps, or failing to follow recommended practice when carrying it out, could lead to misleading or incorrect conclusions. Such conclusions waste time for practitioners, who might profit more from using alternative methods or configurations. They also waste the time of researchers, who make overly strong assumptions based on an arbitrary tuning of evaluation parameters.

We examined 32 recently published papers on fuzz testing (see Table 1) located by perusing top-conference proceedings and other quality venues, and studied their experimental evaluations. We found that no fuzz testing evaluation carries out all of the above steps properly (though some get close). This is bad news in theory, and after carrying out more than 50000 CPU hours of experiments, we believe it is bad news in practice, too. Using AFLFast [6] (as A) and AFL (as baseline B), we carried out a variety of tests of their performance. We chose AFLFast as it was a recent advance over the state of the art; its code was publicly available; and we were confident in our ability to rerun the experiments described by the authors in their own evaluation and expand these experiments by varying parameters that the original experimenters did not. This choice was also driven by the importance of AFL in the literature: 14 out of 32 papers we examined used AFL as a baseline in their evaluation. We targeted three binutils programs (*nm*, *objdump*, and

Cybersecurity science

Why do we want or need this?

- Science helps us understand reality; missing that for security
  - Would like durable understanding – laws (like motion, entropy, etc.)
- Science is successful, current cybersecurity practice is not

Sciences like physics, chemistry, medicine speak about nature

- Security is about human-made systems – science of the *artificial*

Big part of the challenge!

What is applied science?

a) the application of the **scientific method** for achieving **practical goals**, rather than purely for knowledge discovery, *and*

b) the **use of** accumulated **scientific theories**, **knowledge, methods, and techniques** to that end

Louis Pasteur

What are the **practical goals** for security?

Model: Cryptography

- Schneider: “The field of cryptography comes close to exemplifying the kind of science base we seek.”

“Is cryptography scientific?”

Model: Cryptography

- Schneider: “The field of cryptography comes close to exemplifying the kind of science base we seek.”

- Krawczyk: “By its very nature, there is **no (and cannot be) empirical evidence for the security of a design**. Indeed, no concrete measurements or simulations can show that attacks against a cryptographic scheme are not feasible. The only way to do so is to **develop a formal mathematical model and language in which to reason about such schemes**”

The Fundamental Asymmetry

We can observe:

- ✅ Insecurity (attacks succeed, systems fail)

We cannot observe:

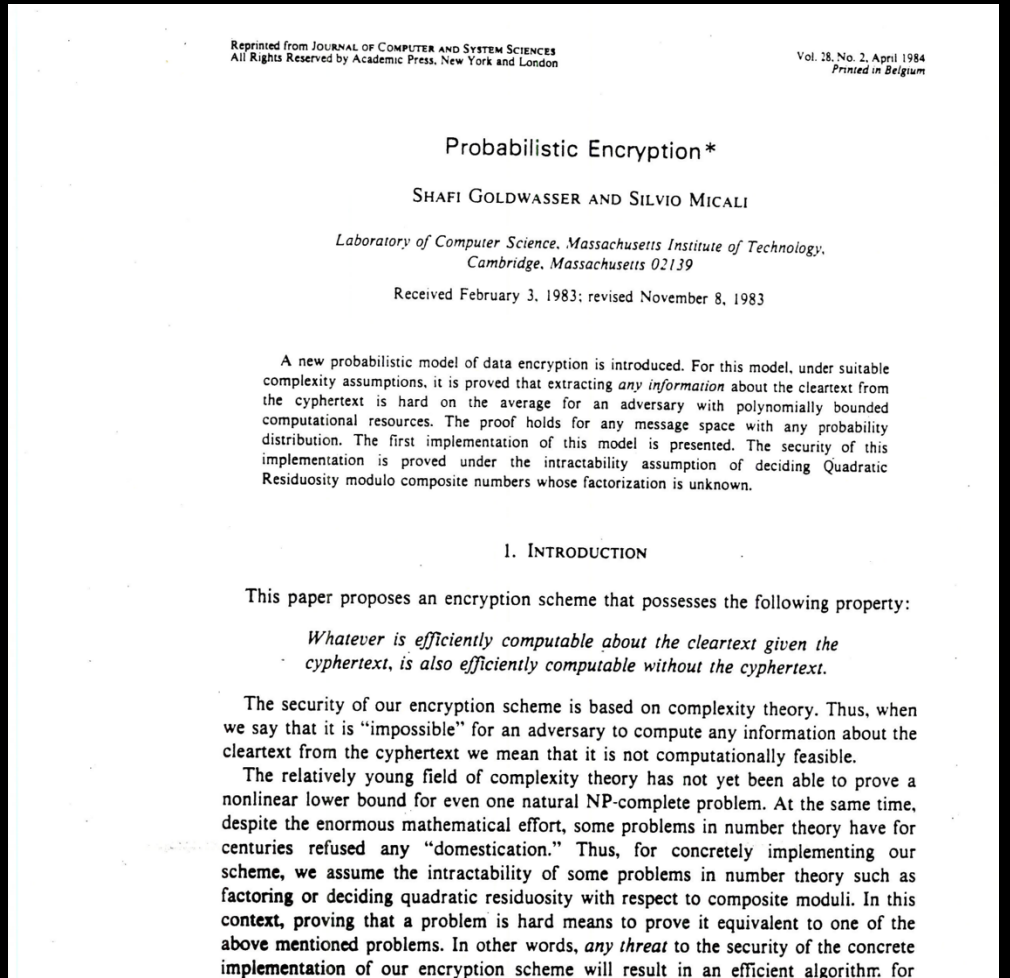
- ❌ Security (absence of attacks ≠ security)

This is **not** a temporary limitation.

It’s a **fundamental** asymmetry.

Provable security

- Cryptography achieved what Schneider wants for all security:
  - **Formal security definitions** (e.g., semantic security)
  - **Adversary models** (what the attacker can do)
  - **Computational assumptions** (e.g., factoring is hard)
  - **Reduction proofs** showing: "break scheme → break assumption"
- **Result:** Mathematically grounded confidence in cryptographic constructions



Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 28, No. 2, April 1984
Printed in Belgium

Probabilistic Encryption*

SHAFI GOLDWASSER AND SILVIO MICALI

Laboratory of Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Received February 3, 1983; revised November 8, 1983

A new probabilistic model of data encryption is introduced. For this model, under suitable complexity assumptions, it is proved that extracting *any information* about the cleartext from the cyphertext is hard on the average for an adversary with polynomially bounded computational resources. The proof holds for any message space with any probability distribution. The first implementation of this model is presented. The security of this implementation is proved under the intractability assumption of deciding Quadratic Residuosity modulo composite numbers whose factorization is unknown.

1. INTRODUCTION

This paper proposes an encryption scheme that possesses the following property:

Whatever is efficiently computable about the cleartext given the cyphertext, is also efficiently computable without the cyphertext.

The security of our encryption scheme is based on complexity theory. Thus, when we say that it is "impossible" for an adversary to compute any information about the cleartext from the cyphertext we mean that it is not computationally feasible.

The relatively young field of complexity theory has not yet been able to prove a nonlinear lower bound for even one natural NP-complete problem. At the same time, despite the enormous mathematical effort, some problems in number theory have for centuries refused any "domestication." Thus, for concretely implementing our scheme, we assume the intractability of some problems in number theory such as factoring or deciding quadratic residuosity with respect to composite moduli. In this context, proving that a problem is hard means to prove it equivalent to one of the above mentioned problems. In other words, *any threat* to the security of the concrete implementation of our encryption scheme will result in an efficient algorithm for

Proofs are only as good as their assumptions

Component	Can Fail When…
Security definition	Doesn’t capture real attack goals
Attacker model	Real attackers have more capabilities
Computational assumption	Assumption turns out false
Reduction	Gap between model and implementation

Deduction vs. Induction

“Deduction in itself is quite powerful as a method of scientific discovery…”

- But with a useful model, you can deduce surprising and useful results
  - Shannon's channel capacity theorem (1948)
  - Gödel's incompleteness theorems (1931)
  - Deductions from Euclidean geometry (e.g., Pythagorean theorem)
- Schneider’s examples
  - Execution monitoring
  - Byzantine fault tolerance

Schneider's view: laws and principles

A science should provide *first principles* that allow developers to predict the consequences of design and implementation choices.

The Cryptography Analogy:

- Cryptography developed *laws* — mathematical foundations
- Information theory gives provable bounds
- **Question:** Can we do the same for security broadly?

Formal methods

- Formally verified replacement for existing AWS authorizer
  - **Deduction**: the implementation matches the specification
  - **Induction**: The specification does what we want it to do …
    - defined by existing reality …
    - confirmed by further deductions!

Formally Verified Cloud-Scale Authorization

Aleks Chakarov, *Amazon*, aleksach@
Jaco Geldenhuys, *Amazon*, jgeldenh@
Matthew Heck, *Amazon*, mcheck@
Michael Hicks, *Amazon*, mwhicks@
Sam Huang, *Amazon*, srhuang@
Georges-Axel Jaloyan, *Amazon*, gjaloyan@
Anjali Joshi, *Amazon*, anjalijs@
K. Rustan M. Leino, *Amazon*, leino@
Mikael Mayer, *Amazon*, mimayere@
Sean McLaughlin, *Amazon*, seanmcl@
Akhilesh Mritunjai, *Amazon*, amritun@
Clement Pit-Claudel, *Amazon*, pitclauc@amazon.ch
Sorawee Porncharoenwase, *Amazon*, soraweep@
Florian Rabe, *Amazon*, florrabe@
Marianna Rapoport, *Amazon*, rapopor@
Giles Reger, *Amazon*, reggiles@
Cody Roux, *Amazon*, codyroux@
Neha Rungta, *Amazon*, rungta@
Robin Salkeld, *Amazon*, salkeldr@
Matthias Schlaipfer, *Amazon*, schlaipf@
Daniel Schoepe, *Amazon*, schoeped@
Johanna Schwartzentruber, *Amazon*, jjsch@
Serdar Tasiran, *Amazon*, tasirans@
Aaron Tomb, *Amazon*, aarotomb@
Emina Torlak, *Amazon*, torlaket@
Jean-Baptiste Tristan, *Amazon*, trjohnb@
Lucas Wagner, *Amazon*, lgwagner@
Michael W. Whalen, *Amazon*, mww@
Remy Willems, *Amazon*, rwillems@
Tongtong Xiang, *Amazon*, ttx@
Tae Joon Byun, *Meta*, taejoon@umn.edu
Joshua Cohen, *Princeton University*, jmc16@cs.princeton.edu
Ruijie Fang, *University of Texas at Austin*, rjf@abstractpredicates.org
Junyoung Jang, *McGill University*, junyoung.jang@mail.mcgill.ca
Jakob Rath, *TU Wien*, jakob.rath@tuwien.ac.at
Hira Taqdees Syeda, *University of Melbourne*, hira.syeda@unimelb.edu.au
Dominik Wagner, *NTU Singapore*, dominik.wagner@cs.ox.ac.uk
Yongwei Yuan, *Purdue University*, yuan311@purdue.edu

***Abstract*—All critical systems must evolve to meet the needs of a growing and diversifying user base. But supporting that evolution is challenging at increasing scale: Maintainers must find a way to ensure that each change does only what is intended, and will not inadvertently change behavior for existing users. This paper presents how we addressed this challenge for the Amazon Web Services (AWS) authorization engine, invoked 1 billion times per second, by using formal verification. Over a period of four years, we built a new authorization engine, one that behaves functionally the same as its predecessor, using the verification-aware programming language Dafny. We can now confidently deploy enhancements and optimizations while maintaining the highest *from the ground up*, and then compile the result to Java. 2) To ensure performance, debuggability, and to gain trust from stakeholders, we needed to generate readable, *idiomatic* Java code, essentially a transliteration of the source Dafny. 3) To ensure that the specification matches the system's actual behavior, we performed *extensive differential and shadow testing* throughout the development process, ultimately comparing against 10^{15} production samples prior to deployment.**

Our approach demonstrates how formal verification can be effectively applied to evolve critical legacy software at scale.

I. Introduction

To control access to their data and resources, AWS cus-

Key challenge: Specifying security

To know if a system is secure, we have to

- Develop a model for it
- Describe what the model should (and should not) do
- Prove the model satisfies our description
- Establish that the model, and its assumptions, represent reality

What do security specifications look like?

Schneider: Hyperproperties

Cybersecurity practice: Anti-patterns

- Unfalsifiable claims
- Confusing sufficient with necessary
- Can never argue anything *out*
- Implicit assumptions
- Data sparsity

Unfalsifiable Claims

Claim: “You must do X to be secure”

To prove this wrong, you need:

1. Something that doesn’t do X
2. That is provably secure

But you can **never prove something is secure.**

Therefore: **Necessity claims are unfalsifiable.**

Confusing Sufficient for Necessary

Sufficient: X is enough to achieve Y

Necessary: You can't achieve Y without X

In security, we often confuse these:

- "Avoiding password reuse is *sufficient* to counter some attacks"
- Gets interpreted as: "Avoiding password reuse is *necessary* for security"
- But it's impossible to achieve across 100 accounts!

The Password Portfolio Impossibility

Standard advice:

1. Passwords should be random and strong (~40 bits)
2. Never reuse passwords across accounts

For N = 100 accounts:

$N \times \log_2(S) + \log_2(N!) = 4{,}000 + 524 =$ **4,524 random bits**

Equivalent to memorizing:

- 1,361 digits of pi
- Order of 17 shuffled card decks

Implicit Assumptions

Morris & Thompson (1979): Password security paper

- Made reasonable assumptions for 1979
- Conclusions persisted long after assumptions changed

When assumptions are implicit:

- Can’t assess if conclusions still hold
- Can’t argue changed circumstances
- Old advice becomes immune to critique

The One-Sided Ratchet

Science advances through self-correction:

Wrong ideas get falsified and abandoned.

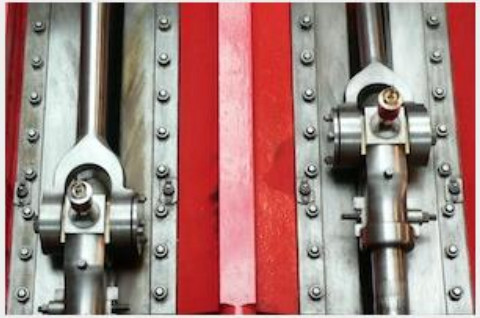
In security:

- ✅ New attacks argue countermeasures **IN**
- ❌ Nothing argues countermeasures **OUT**

Result: A ratchet that only goes one direction

- Requirements accumulate endlessly
- Resources spread across ever more defenses
- Old measures become “received wisdom”

About Artifact Evaluation



In 2011, ESEC/FSE initiated a novel experiment for a major software conference: giving authors the opportunity to submit for evaluation any artifacts that accompany their papers. A similar experiment has since run successfully for several more conferences. This document describes the goals and general mechanics of this process.

If you're just looking for the packaging guidelines, go directly to them.

The rest of this document contains general guidelines about artifact evaluation.
*Individual conferences are **welcome and encouraged** to copy this prose to explain the goals, process, and design to their communities.*

To make things clear to conferences:

Data Sparsity Problem

Attack frequency varies enormously:

- Password Guessing: 10^{-2} to 10^{1} per user
- Spam: 10^{-5} to 10^{-3}
- Intrusion Detection: 10^{-7} to 10^{-5}
- DoS: 10^{-10} to 10^{-7}

When data is sparse, iterative feedback is harder.

Obscuring/denying uncertainty is harmful.