2017

Accountable Algorithms

Joel R. Reidenberg
Fordham University School of Law, jreidenberg@law.fordham.edu

ARTICLE

ACCOUNTABLE ALGORITHMS

JOSHUA A. KROLL, JOANNA HUEY, SOLON BAROCAS, EDWARD W. FELTEN, JOEL R. REIDENBERG, DAVID G. ROBINSON & HARLAN YU†

Many important decisions historically made by people are now made by computers. Algorithms count votes, approve loan and credit card applications, target citizens or neighborhoods for police scrutiny, select taxpayers for IRS audit, grant or deny immigration visas, and more.

The accountability mechanisms and legal standards that govern such decision processes have not kept pace with technology. The tools currently available to policymakers, legislators, and courts were developed to oversee human decisionmakers and often fail when applied to computers instead. For example, how do you judge the intent of a piece of software? Because automated decision systems can return potentially incorrect, unjustified, or unfair results, additional approaches are needed to make such systems accountable and governable. This Article reveals a new technological toolkit to verify that automated decisions comply with key standards of legal fairness.

We challenge the dominant position in the legal literature that transparency will solve these problems. Disclosure of source code is often neither necessary (because of alternative techniques from computer science) nor sufficient (because of the issues analyzing code) to demonstrate the fairness of a process. Furthermore, transparency

† Respectively, Affiliate, Center for Information Technology Policy, Princeton; Associate Director, Center for Information Technology Policy, Princeton; Post Doctoral Research Associate, Princeton; Robert E. Kahn Professor of Computer Science and Public Affairs, Princeton; Stanley D. and Nikki Waxberg Chair in Law, Fordham Law School; Principal, Upturn, and Visiting Fellow, Information Society Project, Yale Law School; Principal, Upturn, and Fellow, Stanford Center for Internet and Society. For helpful comments, the authors are very grateful to participants at the Berkeley Privacy Law Scholars Conference and at the NYU School of Law conference on "Accountability and Algorithms." Research on this Article was supported in part by NSF Award DGE-1148900 and a Fordham Faculty Fellowship.

may be undesirable, such as when it discloses private information or permits tax cheats or terrorists to game the systems determining audits or security screening.

The central issue is how to assure the interests of citizens, and society as a whole, in making these processes more accountable. This Article argues that technology is creating new opportunities—subtler and more flexible than total transparency—to design decisionmaking algorithms so that they better align with legal and policy objectives. Doing so will improve not only the current governance of automated decisions, but also—in certain cases—the governance of decisionmaking in general. The implicit (or explicit) biases of human decisionmakers can be difficult to find and root out, but we can peer into the "brain" of an algorithm: computational processes and purpose specifications can be declared prior to use and verified afterward.

The technological tools introduced in this Article apply widely. They can be used in designing decisionmaking processes from both the private and public sectors, and they can be tailored to verify different characteristics as desired by decisionmakers, regulators, or the public. By forcing a more careful consideration of the effects of decision rules, they also engender policy discussions and closer looks at legal standards. As such, these tools have far-reaching implications throughout law and society.

Part I of this Article provides an accessible and concise introduction to foundational computer science techniques that can be used to verify and demonstrate compliance with key standards of legal fairness for automated decisions without revealing key attributes of the decisions or the processes by which the decisions were reached. Part II then describes how these techniques can assure that decisions are made with the key governance attribute of procedural regularity, meaning that decisions are made under an announced set of rules consistently applied in each case. We demonstrate how this approach could be used to redesign and resolve issues with the State Department's diversity visa lottery. In Part III, we go further and explore how other computational techniques can assure that automated decisions preserve fidelity to substantive legal and policy choices. We show how these tools may be used to assure that certain kinds of unjust discrimination are avoided and that automated decision processes behave in ways that comport with the social or legal standards that govern the decision. We also show how automated decisionmaking may even complicate existing doctrines of disparate treatment and disparate impact, and we discuss some recent computer science work on detecting and removing discrimination in algorithms, especially in the context of big data and machine learning. And lastly, in Part IV, we propose an agenda to further synergistic collaboration between computer science, law, and policy to advance the design of automated decision processes for accountability.

INTRODUCTION

Many important decisions that were historically made by people are now made by computer systems[1]: votes are counted; voter rolls are purged; loan and credit card applications are approved;[2] welfare and financial aid decisions are made;[3] taxpayers are chosen for audits; citizens or neighborhoods are targeted for police scrutiny;[4] air travelers are selected for search;[5] and visas are granted or denied. The efficiency and accuracy of automated decisionmaking ensures that its domain will continue to expand. Even mundane activities now involve complex computerized decisions: everything from cars to home appliances now regularly executes computer code as part of its normal operation.

However, the accountability mechanisms and legal standards that govern decision processes have not kept pace with technology. The tools currently available to policymakers, legislators, and courts were developed primarily to oversee human decisionmakers. Many observers have argued that our current frameworks are not well-adapted for situations in which a potentially incorrect,[6] unjustified,[7] or unfair[8] outcome emerges from a computer. Citizens, and society as a whole, have an interest in making these processes more accountable. If these new inventions are to be made governable, this gap must be bridged.

---

1 In this Article, we use the term "computer system" where others have used the term "algorithm." *See, e.g.*, FRANK PASQUALE, THE BLACK BOX SOCIETY: THE SECRET ALGORITHMS THAT CONTROL MONEY AND INFORMATION (2015). This allows us to separate the concept of a computerized decision from the actual machine that effects it. *See infra* note 14 for a more detailed explanation.

2 *See, e.g.*, *Calyx - More Than Just an LOS*, CALYX SOFTWARE (Mar. 2013), http://www.calyxsoftware.com/company/newsletters/13-03.html [https://perma.cc/E93L-8UGD] (noting that Calyx offers clients an automated underwriting system to vet loan applications for approval against predetermined guidelines).

3 *See* Virginia Eubanks, *Caseworkers v. Computers*, POPTECH (Dec. 11, 2013, 3:10 PM), http://virginiaeubanks.wordpress.com/2013/12/11/caseworkers-vs-computers [https://perma.cc/37VG-GQC6] (describing and critiquing several states' efforts to automate welfare eligibility determinations).

4 *See* DAVID ROBINSON, HARLAN YU & AARON RIEKE, CIVIL RIGHTS, BIG DATA, AND OUR ALGORITHMIC FUTURE 18-19 (2014), http://bigdata.fairness.io/wp-content/uploads/2014/11/Civil_Rights_Big_Data_and_Our_Algorithmic-Future_v1.1.pdf [https://perma.cc/UL3G-3MQ7] (describing the Chicago Police Department's "'Custom Notification Program,' which sends police (or sometimes mails letters) to peoples' homes to offer social services and a tailored warning").

5 *See* Notice of Modified Privacy Act System of Records, 78 Fed. Reg. 55,270, 55,271 (Sept. 10, 2013) ("[T]he passenger prescreening computer system will conduct risk-based analysis of passenger data. . . . TSA will then review this information using intelligence-driven, risk-based analysis to determine whether individual passengers will receive expedited, standard, or enhanced screening . . . .").

6 *See* Danielle Keats Citron, *Technological Due Process*, 85 WASH. U. L. REV. 1249, 1256 (2008) (describing systemic errors in the automated eligibility determinations for federal benefits programs).

7 *See id.* at 1256-57 (noting the "crudeness" of algorithms designed to identify potential terrorists that yield a high rate of false positives).

8 *See* Solon Barocas & Andrew D. Selbst, *Big Data's Disparate Impact*, 104 CALIF. L. REV. 671, 677 (2016) ("[D]ata mining holds the potential to unduly discount members of legally protected classes and to place them at systematic relative disadvantage.").

In this Article, we describe how authorities can demonstrate—and how the public at large and oversight bodies can verify—that automated decisions comply with key standards of legal fairness. We consider two approaches: ex ante approaches aiming to establish that the decision process works as expected (which are commonly studied by technologists and computer scientists), and ex post approaches once decisions have been made, such as review and oversight (which are common in existing governance structures). Our proposals aim to use the tools of the first approach to guarantee that the second approach can function effectively. Specifically, we describe how technical tools for verifying the correctness of computer systems can be used to ensure that appropriate evidence exists for later oversight.

We begin with an accessible and concise introduction to the computer science concepts on which our argument relies, drawn from the fields of software verification, testing, and cryptography. Our argument builds on the fact that technologists can and do verify for themselves that software systems work in accordance with known designs. No computer system is built and deployed in the world shrouded in total mystery.[9] While we do not advocate any specific liability regime for the creators of computer systems, we outline the range of tools that computer scientists and other technologists already use, and show how those tools can ensure that a system meets specific policy goals. In particular, while some of these tools provide assurances only to the system's designer or operator, other established methods could be leveraged to convince a broader audience, including regulators or even the general public.

The tools available during the design and construction of a computer system are far more powerful and expressive than those that can be bolted on to an existing system after one has been built. We argue that, in many instances, designing a system for accountability can enable stakeholders to reach accountability goals that could not be achieved by imposing new transparency requirements on existing system designs.

We show that computer systems can be designed to prove to oversight authorities and the public that decisions were made under an announced set of rules consistently applied in each case, a condition we call *procedural regularity*. The techniques we describe to ensure procedural regularity can be extended to demonstrate adherence to certain kinds of substantive policy choices, such as blindness to a particular attribute (e.g., race in credit underwriting). Procedural regularity ensures that a decision was made using consistently applied standards

9 Although some machine learning systems produce results that are difficult to predict in advance and well beyond traditional interpretation, the choice to field such a system instead of one which can be interpreted and governed is itself a decision about the system's design. While we do not advocate that any approach should be forbidden for any specific problem, we aim to show that advanced tools exist that provide the desired functionality while also permitting oversight and review.

and practices. It does not, however, guarantee that such practices are themselves good policy. Ensuring that a decision procedure is well justified or relies on sound reasoning is a separate challenge from achieving procedural regularity. While procedural regularity is a well-understood and generally desirable property for automated and nonautomated governance systems alike, it is merely one principle around which we can investigate a system's fairness.

It is common, for example, to ask whether a computer system avoids certain kinds of unjust discrimination, even when such systems are blind to certain attributes (e.g., gender in automated hiring decisions). We later expand our discussion and show how emerging computational techniques can assure that automated decisions satisfy other notions of fairness that are not merely procedural, but actively consider a system's effects. We describe in particular detail techniques for avoiding discrimination, even in machine learning systems that derive their decision rules from data rather than from code written by a programmer. And finally, we propose next steps to further the emerging and critically important collaboration between computer scientists and policymakers.

Legal scholars have argued for twenty years that automated processing requires more transparency,[10] but it is far from obvious what form such transparency should take. Perhaps the most obvious approach is to disclose a system's source code, but this is at best a partial solution to the problem of accountability for automated decisions. The source code of computer systems is illegible to nonexperts. In fact, even experts often struggle to understand what software code will do, as inspecting source code is a very limited way of predicting how a computer program will behave.[11] Machine learning, one increasingly popular approach to automated decisionmaking, is particularly ill-suited to source code analysis because it involves situations where the decisional rule itself emerges automatically from the specific data under analysis, sometimes in ways that no human can explain.[12] In this case, source code alone teaches a reviewer very little, since the code only exposes the machine learning method used and not the data-driven decision rule.

---

10 *See, e.g.*, Citron, *supra* note 6, at 1253 (describing automated decisionmaking as "adjudicat[ion] in secret"); Paul Schwartz, *Data Processing and Government Administration: The Failure of the American Legal Response to the Computer*, 43 HASTINGS L.J. 1321, 1323-25 (1992) ("So long as government bureaucracy relies on the technical treatment of personal information, the law must pay attention to the structure of data processing . . . . There are three essential elements to this response: structuring transparent data processing systems; granting limited procedural and substantive rights . . . and creating independent governmental monitoring of data processing systems." (emphasis omitted)).

11 *See infra* subsection I.A.1 (discussing static analysis).

12 *See* Stanford Univ., *Machine Learning*, COURSERA, https://www.coursera.org/learn/machine-learning/home/info [https://perma.cc/L7KF-CDY4] ("Machine learning is the science of getting computers to act without being explicitly programmed.").

Moreover, in many of the instances that people care about, full transparency will not be possible. The process for deciding which tax returns to audit, or whom to pull aside for secondary security screening at the airport, may need to be partly opaque to prevent tax cheats or terrorists from gaming the system. When the decision being regulated is a commercial one, such as an offer of credit, transparency may be undesirable because it defeats the legitimate protection of consumer data, commercial proprietary information, or trade secrets. Finally, when an explanation of how a rule operates requires disclosing the data under analysis and those data are private or sensitive (e.g., in adjudicating a commercial offer of credit, a lender reviews detailed financial information about the applicant), disclosure of the data may be undesirable or even legally barred.

Furthermore, making the rule transparent—whether through source code disclosure or otherwise—may still fail to resolve the concerns of many participants. No matter how much transparency surrounds a rule, people can still wonder whether the disclosed rule was actually used to reach a decision in their own cases. Particularly where an element of randomness is involved in the process, a person audited or patted down may wonder: was I really chosen by the rule, or has some bureaucrat singled me out on a whim? But full disclosure of how particular decisions were reached is often unattractive because the decisions themselves often incorporate sensitive health, financial, or other private information either as input, output, or both (for example, an individual's tax audit status may be sensitive or protected on its own, but it may also imply details about that individual's financial data).

Even full disclosure of a decision's provenance to that decision's subject can be problematic. Most individuals are ill-equipped to review how computerized decisions are made, even if those decisions are reached transparently. Further, the purpose of computer-mediated decisionmaking is to bring decisions an element of scale, where the same rules are ostensibly applied to a large number of individual cases or are applied extremely quickly. Individuals auditing their own decisions (or experts assisting them) would be both inundated with the need to review the rules applied to them and often able to generalize their conclusions to the results of others, raising the same disclosure concerns described above. That is, while transparency of a rule makes reviewing the basis of decisions more possible, it is not a substitute for individualized review of particular decisions.[13]

13 Even when experts can pool investigative effort across many decisions, there is no guarantee that the basis for decisions will be interpretable or that problems of fairness or even overt special treatment for certain people will be discovered. Further, a regime based on individuals auditing their own decisions cannot adequately address departures from an established rule, which favor the individual auditing her own outcome, or properties of the rule, which can only be examined across individuals (such as nondiscrimination).

Fortunately, technology is creating new opportunities—more subtle and flexible than total transparency—to make automated decisionmaking more accountable to legal and policy objectives. Although the current governance of automated decisionmaking is underdeveloped, computerized processes can be designed for governance and accountability. Doing so will improve not only the current governance of computer systems, but also—in certain cases—the governance of decisionmaking in general.

This Article argues that in order for a computer system to function in an accountable way—either while operating an important civic process or merely engaging in routine commerce—accountability must be part of the system's design from the start. Designers of such systems, and the nontechnical stakeholders who often oversee or control system design, must begin with oversight and accountability in mind. We offer examples of currently available tools that could aid in that design, as well as suggestions for dealing with the apparent mismatch between policy ambiguity and technical precision.

In Part I of this Article, we provide an accessible introduction to how computer scientists build and evaluate computer systems and the software and algorithms[14] that comprise them. In particular, we describe how computer

---

14 In this Article, we limit our use of the word "algorithm" to its usage in computer science, where it refers to a well-defined set of steps for accomplishing a certain goal. In other contexts, where other authors have used the term "algorithm," we describe "automated decision processes" reflecting "decision policies" implemented by pieces of "software," all comprising "computer systems." Our adoption of the phrase "computer systems" was suggested by (and originally due to) Helen Nissenbaum, and we are grateful for the precision it provides. *See generally* Batya Friedman & Helen Nissenbaum, *Bias in Computer Systems*, 14 ACM TRANSACTIONS ON INFO. SYSTEMS 330 (1996).

The term "algorithm" is assigned disparate technical meanings in the literatures of computer science and other fields. The computer scientist Donald Knuth famously defined algorithms as separate from mathematical formulae in that (1) they must "always terminate after a finite number of steps;" (2) "[e]ach step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case;" (3) input to the algorithm is "quantities that are given to it initially before the algorithm begins;" (4) an algorithm's output is "quantities that have a specified relation to the inputs;" and (5) the operations to be performed in the algorithm "must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper." 1 DONALD E. KNUTH, THE ART OF COMPUTER PROGRAMMING: FUNDAMENTAL ALGORITHMS 4-6 (1968). Similarly and more simply, a widely used computer science textbook defines an algorithm as "any well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output*." THOMAS H. CORMEN ET AL., INTRODUCTION TO ALGORITHMS 10 (2d ed. 2001).

By contrast, communications scholar Christian Sandvig says that "'algorithm' refers to the overall process" by which some human actor uses a computer to do something, including decisions made by humans as to what the computer should do, choices made during implementation, and even choices about how algorithms are represented and marketed to the public. Christian Sandvig, *Seeing the Sort: The Aesthetic and Industrial Defense of "The Algorithm,"* MEDIA-N, http://median.newmediacaucus.org/art-infrastructures-information/seeing-the-sort-the-aesthetic-and-industrial-defense-of-the-algorithm [https://perma.cc/29E4-S44S]. Sandvig argues that even algorithms as simple as sorting "have their own public relations" and are inherently human in their decisions. *Id.*

scientists evaluate a program to verify that it has desired properties and discuss the value of randomness in the construction of many computer systems. We characterize what sorts of properties of a computer system can be tested and describe one of the fundamental truths of computer science—that there are some properties of computer systems which cannot be tested completely. We observe that computer systems fielded in the real world are (or at least should be) tested regularly during creation, deployment, and operation, merely to establish that they are actually functional.

Part II examines how to design computer systems for procedural regularity, a key governance principle enshrined in law and public policy in many societies. We consider how participants, decision subjects, and observers can be assured that each individual decision was made according to the same procedure—for example, how observers can be assured that the decisionmaker is not choosing outcomes on a whim while merely claiming to follow an announced rule. We describe why mere disclosure of a piece of source code can be impractical or insufficient for these ends. Indeed, without full transparency—including source code, input data, and the full operating environment of the software—even the disclosure of audit logs showing what a program did while it was running provides no guarantee that the disclosed information actually reflects a computer system's behavior.[15] In order to move beyond the need for full transparency, we focus on tools that can communicate partial information about secret processes, so that accountability and oversight continue to function even when policy interests, personal privacy, trade secrets, or other concerns protect a computer system, a piece of software, its inputs, its outputs, or its environment from disclosure. Putting it all together, we provide an illustrative example of how to redesign an existing, legally mandated automated decisionmaking system—the State Department's Diversity Visa Lottery—so that it is provably accountable.

Another communications scholar, Nicholas Diakopoulos, defines algorithms in the narrow sense as "a series of steps undertaken in order to solve a particular problem or accomplish a defined outcome," but also considers them in the broad sense, saying that "algorithms can arguably make mistakes and operate with biases," which does not make sense for the narrower technical definition. Nicholas Diakopoulos, *Algorithmic Accountability: Journalistic Investigation of Computational Power Structures*, 3 DIGITAL JOURNALISM 398, 398, 400 (2015). This confusion is common in much of the literature on algorithms and accountability, which we describe throughout this Article. To avoid confusion, this Article adopts the precise definition of the word "algorithm" from computer science and, following Friedman and Nissenbaum, refers to the broader concept of an automated system deployed in a social or human context as a "computer system."

15 The environment of a computer system includes anything it might interact with. For example, an outside observer will need to know what other software was running on a particular computer to ensure that nothing modified the behavior of the disclosed program. Some programs also observe (and change their behavior based on) the state of the computer they are running on (such as which files were or were not present or what other programs were running), the time they were run, or even the configuration of hardware on the system on which they were run.

Part III considers the broader question of how to assess a computer system's compliance with policy principles that go beyond procedural regularity. These broader properties include determining whether automated decision systems treat people (including protected groups) in ways that comport with the social or legal standards that govern the decision being made.[16] This broadening raises the issue of translating a policy principle into a property of the system. Certain substantive policy choices translate easily: for example, prespecified rules such as blindness to a sensitive attribute.[17] Defining other policy objectives, such as a general notion of nondiscrimination, however, is a more complicated and fraught affair, particularly when systems rely on machine learning rather than decision rules explicitly predetermined by humans. We explore in particular the discriminatory effect that automated decisionmaking can have, noting real-world examples of newfound risks and describing some system properties that may align with policy goals. Finally, we observe how automated decisionmaking may complicate the existing doctrine of disparate treatment and disparate impact.

Part IV concludes by calling for increased collaboration between computer scientists and policymakers to develop and apply technical tools for the governance of computer systems. Given the ever-widening reach of automated decisions, computer scientists need to understand the policy challenges of oversight, and policymakers need to understand where new and emerging software tools can help address those challenges. We offer recommendations for bridging the gap between technologists' desire for specificity and the policy process's need for ambiguity. As a first step, we urge policymakers to recognize that accountability is feasible even when the details of a computer system are not fully known or must be kept secret. We also argue that the ambiguities, contradictions, and uncertainties of the policy process need not discourage computer scientists from engaging constructively in it.

I. How Computer Scientists Build and Evaluate Software

Fundamentally, computers are general purpose machines that can be programmed to do any computational task, though they lack the desirable specificity and limitations of physical devices.[18] Engineers often seek strong

16 This type of evaluation depends upon having already verified procedural regularity: if it cannot be determined that a particular algorithm was used to make a decision, it is fruitless to try to verify properties of that algorithm.

17 A concrete example would be the requirement that a decision only account for certain information for certain purposes, as in a system for screening job applicants that is allowed to take the gender of applicants as input, but only for the purpose of keeping informational statistics and not for making screening decisions.

18 For example, hydraulically operated control surfaces in a vehicle will telegraph resistance to the operator when they are close to a dangerous configuration, but the same controls operated by a

digital evidence that a computer system is working as intended. Such evidence may be persuasive for the system's creator or operator, for a predesignated group of receivers such as an oversight authority, or for the public at large. In many cases, systems are carefully evaluated and tested before they make it to the real world. Evidence that is convincing to the public and sufficiently nonsensitive to be disclosed widely is the most effective and desirable for ensuring accountability.

In this Part, we examine how computer scientists think about software assurance, how software is built and tested in the software industry, and what tools are available to get assurances about an individual piece of software or a large computer system. Thus, this Part provides a brief and accessible map of key concepts and offers some insight into how computer scientists think about and approach these challenges.

A. *Assessing Computer Systems*

In general, a computer program is something that takes a set of inputs and produces a set of outputs. All too often, programs fail to work as their authors intended because the programs have bugs or make assumptions about the input data that are not always true. Programmers often structure or design programs with an eye toward evaluation and testing in order to avoid or minimize these pitfalls.[19] Many respected and popular approaches to software engineering are based on the idea that code should be written in ways that make it easier to analyze.[20] For example, the programmer can:

- Organize the code into modules that can be evaluated separately and then combined.[21]
- Test these modules for proper functionality both individually and in groups, possibly even testing the entire computer system end-to-end. Such testing generally involves writing test cases, or expected scenarios in which each module will run, and may involve running

computer can omit feedback, allowing the computer to request configurations of actuators that are beyond their tolerances. This is a problem especially in the design of robotic arms and fly-by-wire systems for aircraft.

19 *See* ANDREW HUNT & DAVID THOMAS, THE PRAGMATIC PROGRAMMER: FROM JOURNEYMAN TO MASTER 196 (2000) (describing a "[c]ulture of [t]esting" in which programmers should plan on testing since a "little forethought can go a long way toward minimizing maintenance costs and help-desk calls"); *see also id.* at 41 (advocating for an "orthogonally designed and implemented system" because it "is easier to test").

20 In particular, Test Driven Development (TDD) is a software engineering methodology practiced by many major software companies. For a general description of how TDD integrates automated testing into software design, see KENT BECK, TEST-DRIVEN DEVELOPMENT: BY EXAMPLE (2003).

21 *See* HUNT & THOMAS, *supra* note 19, at 34-43.

test cases each time the software is changed to avoid introducing new bugs or taking away functionality unintentionally.[22]

- Annotate the code with *assertions*, simple statements about the code that describe error conditions under which the program should crash immediately. Assertions are intended to be true if the program is running as expected. They become false when something is amiss and cause the program to crash (with an error message) rather than continuing in an errant state. In this way, assertions are a special kind of program error—a point at which a piece of software considers its internal state and its environment, and stops if these do not match what had been assumed by the program's author.[23]
- Provide a detailed description specifying the program's behavior along with a machine-checkable proof that the code satisfies this specification. This differs from using assertions in that the proof guarantees ahead of time that the program will work as intended in all cases (or, equivalently, that an assertion of the facts covered by the proof will never fail to be true). When feasible, this approach is the most helpful thing a programmer can do to facilitate testing because it can provide real proof (rather than just circumstantial evidence or evidence linked to a particular point in a program's execution, as with assertions) that the whole program operates as expected.[24]

These techniques are illustrative examples from a larger toolbox. Testing and verification of software and the development of tools to facilitate it comprise a rich and active subfield of computer science research. A thriving

---

22 *See* STEVE MCCONNELL, CODE COMPLETE: A PRACTICAL HANDBOOK OF SOFTWARE CONSTRUCTION 528 (2d ed. 2004) (explaining regression testing after each change in code); *see also id.* at 499-533 (discussing testing and test cases).

23 This technique derives from 1 HERMAN H. GOLDSTINE & JOHN VON NEUMANN, PLANNING AND CODING OF PROBLEMS FOR AN ELECTRONIC COMPUTING INSTRUMENT (1947), but it is now a widely used technique. For a historical perspective, see Lori A. Clarke & David S. Rosenblum, *A Historical Perspective on Runtime Assertion Checking in Software Development*, 31 ACM SIGSOFT SOFTWARE ENGINEERING NOTES 25 (2006).

24 A simple example is a technique called *model checking*, which is usually applied to computer hardware designs, in which the property desired and the hardware or program are represented as logical formulae and an automated tool performs an exhaustive search (i.e., tries all possible inputs) to check whether those formulae are inconsistent. *See generally* EDMUND M. CLARKE, JR. ET AL., MODEL CHECKING (1999). An even simpler example comes from the concept of *types* in programming languages, which associate the data values on which the program operates into descriptive classes and provide rules for how those classes should interact. For example, it should not be possible to add mathematically a number like "42" to a string of text like "Hello, World!" Because both kinds of data are represented inside the computer as bits and bytes, without a type system, the computer would be free to try executing this nonsensical behavior, which might lead to bugs. Type systems can help programmers avoid mistakes and express extremely complex relationships among the data processed by the program. For a more thorough explanation of type systems and model checking, see BENJAMIN C. PIERCE, TYPES AND PROGRAMMING LANGUAGES (2002).

industry builds tools to assist in the development of software, and there is a constant debate about the practice of software engineering in the technology industry more broadly. Below, we give a brief taxonomy of this area, as well as some examples of tools, expectable results, and limitations of each category. We conclude this Section by describing why testing code after it has been written, however extensively, cannot provide true assurance of how the system works, because any analysis of an existing computer program is inherently and fundamentally incomplete. This incompleteness implies that observers can never be certain that a computer system has a desired property unless that system has been designed to guarantee that property.

When technologists evaluate computer systems, they attempt to establish *invariants*, or facts about a program's behavior that are always true regardless of a program's internal state or the input data the program receives.[25] Invariants can cover details as small as the behavior of a single line of code but can also express complex properties of entire programs, such as which users have access to which data or under what conditions the program can crash. By structuring code modules and programs in a way that makes it easy to establish simple invariants, it is possible to build an entire computer system for which important desirable invariants can be proved.[26] Together, the set of invariants that a program should have make up its *specification*.[27]

---

25 *See* HUNT & THOMAS, *supra* note 19, at 110; C.A.R. Hoare, *An Axiomatic Basis for Computer Programming*, COMM. ACM, Oct. 1969, at 576, 576, 577-80 (providing a foundational examination of how to prove properties of a program).

26 *See* Hoare, *supra* note 25, at 576-80.

27 Specifications can be formal and written in a *specification language*, in which case they are rather like computer programs unto themselves. For example, the early models of core internet technology were written in a language called LOTOS, built for that purpose. *See* Tommaso Bolognesi & Ed Brinksma, *Introduction to the ISO Specification Language LOTOS*, 14 COMPUTER NETWORKS & ISDN SYSTEMS 25, 25 (1987) ("LOTOS is a specification language that has been specifically developed for the formal description of the OSI (Open Systems Interconnection) architecture . . . ."). Other common specification languages in practical use include Z and UML. It is even possible to build an executable program by compiling such a specification into a programming language or directly into machine code, an area of computer science research known as *program synthesis*. *See* Zohar Manna & Richard Waldinger, *A Deductive Approach to Program Synthesis*, 2 ACM TRANSACTIONS ON PROGRAMMING LANGUAGES & SYSTEMS 90, 90 (1980) ("Program synthesis is the systematic derivation of a program from a given specification."). Research has shown that when the language in which a program or specification is written more closely matches a human-readable description of the program's design goals, programs are written with fewer bugs. *See* Michael C. McFarland et al., *The High-Level Synthesis of Digital Systems*, 78 PROC. IEEE 301 (1990) (summarizing early "high-level language" oriented program synthesis); *see also* MCCONNELL, *supra* note 22, at 457 (arguing that reducing the complexity of code so that it is more comprehensible to humans increases reliability and reduces errors). Specifications can also be informal and take the form of anything from a mental model of a system in the mind of a programmer to a detailed written document describing all goals and use cases for a system. The world of industrial software development is full of paradigms and best practices for producing specifications and building code that meets them.

Software code is, ultimately, a rigid and exact description of itself: the code both describes and causes the computer's behavior when it runs.[28] In contrast, public policies and laws are characteristically imprecise, often deliberately so.[29] Thus, even when a well-designed piece of software does assure certain properties, there will always remain some room to debate whether those assurances match the requirements of public policy. The methods described in this Article are designed to forward, rather than to foreclose, debates about what laws mean and how they ought to work. Our approach aims to empower the policy process by empowering the policymaker's tools for dealing with ambiguity and lack of precision, namely review and oversight. We wish to show that software does work as described, allowing a reviewer to determine precisely which properties of the software created a particular rule enforced for a particular decision. Further, if a precise specification of a policy does exist, we wish to show that the software which claims to implement that policy in fact does so.

The specification of a system is a critical question for assessment, and system implementers should be prepared to describe the invariants that their system provides. Verification allows the claims of a system's implementer to constitute evidence that the software in question in fact satisfies those claims.[30] Without strong evidence of a computer system's correctness, even the author of that system cannot reliably claim that it will behave according to a desired policy, and no policymaker or overseer should believe such a claim. For example, a medical radiation device with a software control module was approved for use on patients based on the manufacturer's claims, but a subtle bug in the software allowed it to administer unsafe levels of radiation, which resulted in six accidents and three deaths.[31]

Computer scientists evaluate programs using two testing methodologies: (1) static methods, which look at the code without running the program; and (2) dynamic methods, which run the program and assess the outputs for

---

28 *See* DAVID A. PATTERSON & JOHN L. HENNESSY, COMPUTER ORGANIZATION AND DESIGN: THE HARDWARE/SOFTWARE INTERFACE 13-16 (5th ed. 2014) (describing the translation of software code into instructions that can be executed by hardware).

29 *See, e.g.*, Joseph A. Grundfest & A.C. Pritchard, *Statutes with Multiple Personality Disorders: The Value of Ambiguity in Statutory Design and Interpretation*, 54 STAN. L. REV. 627, 628 (2002) (explaining how legislators often obscure the meaning of a statute to allow for multiple interpretations).

30 *See* CARLO GHEZZI ET AL., FUNDAMENTALS OF SOFTWARE ENGINEERING 269-73 (2d ed. 2002).

31 The commission reviewing the accidents determined that overconfidence on the part of engineers and operators led to both a failure to prevent the problem in the first place and a failure to recognize it as a problem even after multiple accidents had occurred. For an overview, see NANCY LEVESON, *Medical Devices: The Therac-25*, *in* SAFEWARE: SYSTEM SAFETY AND COMPUTERS app. (1995), an update of the earlier article, Nancy G. Leveson & Clark S. Turner, *An Investigation of the Therac-25 Accidents*, COMPUTER, July 1993, at 18.

particular inputs or the state of the program as it is running.[32] Dynamic methods can be divided into (a) observational methods in which an analyst can see how the program runs in the field with its natural inputs; and (b) testing methods, which are more powerful, where an analyst chooses inputs and submits them to the program.[33]

1. Static Analysis: Review from Source Code Alone

Reading source code allows an analyst to learn a great deal about how a program works, but it has some major limitations. Code can be complicated or obfuscated, and even expert analysis often misses eventual problems with the behavior of the program. For example, the Heartbleed security flaw was a potentially catastrophic vulnerability for most internet users that was caused by a common programming error—but that error made it through an open source vetting process and then sat unnoticed for two years, even though anyone was free to read and analyze the code during that time.[34] While there exist automated tools for discovering bugs in source code, even best-of-breed commercial solutions designed to discover exactly this class of error did not find the Heartbleed bug because its structure was subtly different from what automated tools had been designed to recognize.[35] This experience underscores how difficult it can be to find small and simple mistakes. More complex errors evade scrutiny even more easily.

Further, static methods, on their own, say nothing about how a program interacts with its environment.[36] A program that examines any sort of external data, even the time of day, may have different behavior when run in different contexts. For example, it has long been programming practice to use the time of day as the starting value for a chaotic function designed to produce random numbers in programs that do statistical sampling.[37] Such programs naturally choose a different sample of data based on the time of day when

32 *See* GARY MCGRAW, SOFTWARE SECURITY: BUILDING SECURITY IN 179 (2006) ("Static analysis tools can vet software code, either in source or binary form, in an attempt to identify common implementation-level bugs such as buffer overflows. Dynamic analysis tools can observe a system as it executes.").

33 *See id.* ("Dynamic analysis tools . . . can submit malformed, malicious, and random data to a system's entry points in an attempt to uncover faults—a process commonly referred to as fuzzing.").

34 Edward W. Felten & Joshua A. Kroll, *Heartbleed Shows Government Must Lead on Internet Security*, SCI. AM. (Apr. 16, 2014), http://www.scientificamerican.com/article/heartbleed-shows-government-must-lead-on-internet-security [https://perma.cc/QLN4-TUQM].

35 *Id.*

36 *See* MCGRAW, *supra* note 32, at 201 (describing software security problems related to malicious user inputs, register settings, environment variables, file contents, network configuration, and other outside factors).

37 This practice dates back at least as far as the 1989 standard for the C programming language, ANSI X3.159-1989.

they were started, meaning that their output cannot be reproduced unless the time is explicitly represented as an input to the program.

Depending on the technology used to implement a program, static analysis might lead to incomplete or incorrect conclusions simply because it fails to consider the *dependencies*—that is, the other software that a given program needs in order to operate correctly.[38] For some technologies, the same line of code can have radically different meanings based on the version of even a single dependency.[39] Because of this, it is necessary for static analysis to cover a large portion of any system, and to include at least some dynamic information about how a program will be run.

Within limits, static methods can be very useful in establishing facts about a program, such as the nature of the data it takes in, the kind of output it can produce, the general shape of the program, and the technologies involved in the program's implementation.[40] In particular, static analysis can reveal the kinds of inputs that might cause the program to behave in particular ways.[41] Analysts can use this insight to test the program on different types of inputs. Advanced analysis can, in some cases, determine aspects of a program's behavior and establish program *invariants*, or facts about the program's

---

38 *See, e.g.*, *Managing Software Dependencies*, GOV.UK SERVICE MANUAL, https://www.gov.uk/service-manual/technology/managing-software-dependencies [https://perma.cc/3BA8-CXED] ("Most digital services will rely on some third party code from other software to work properly. This is called a dependency.").

39 For example, it is common to reuse "library" code, which provides generic functionality and can be shared across many programs. GHEZZI ET AL., *supra* note 30. Library functions can be very different from version to version, meaning that running a program with a different version of the same library can radically change its behavior. It can even change programs that fail to run at all into running, working programs. Because Microsoft Windows refers to its system libraries as "Dynamic Linked Libraries," developers often call this "DLL Hell." Rick Anderson, *The End of DLL Hell*, MICROSOFT (Jan. 11, 2000), https://web.archive.org/web/20081022192553/http://msdn.microsoft.com/en-us/library/ms811694.aspx [https://perma.cc/WNQ2-AKW9]. Further, in some programming languages, such as PHP, the meaning of certain statements is configurable. *See The Configuration File*, PHP, http://php.net/manual/en/configuration.file.php) [https://perma.cc/9LFC-62D9] (describing configuration of PHP).

40 *See* MCGRAW, *supra* note 32, at 106, 109 (describing both the benefits and limitations of static code analysis).

41 In programming languages, the most basic structure for expressing behavior that depends on a value is a *conditional* statement, often written as *if X then Y else Z*. A conditional statement will execute certain code (*Y*) if the condition (*X*) is met and different code (*Z*) if the condition is not met. Static analysis can reveal where a program has conditional logic, even if it may not always be able to determine which branch of the conditional logic will actually be executed. For example, static analysis of conditional logic might show an analyst that a program behaves one way for inputs less than a threshold and another way otherwise, or that it behaves differently in some particular special case. Generalizing this analysis can allow analysts to break the inputs of a program into classes and evaluate how the program behaves on each class. For an overview of logical constructs in computer programs, see HAROLD ABELSON ET AL., STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS (2d ed. 1996), which describes the elements of programs and how they can be combined and manipulated, and explains conditional expressions and predicates in section 1.1.6.

behavior which are true regardless of what input data the program receives.[42] Programs that are specially designed to take advantage of more advanced analysis techniques can enable an analyst to use static methods to prove formally complex invariants about the program's behavior.[43] On the simplest level, some programming languages are designed to prevent certain classes of mistakes. For example, some are designed in such a way that it is impossible to make the mistake that caused the Heartbleed bug.[44] These techniques have also been deployed in the aviation industry, for example, to ensure that the software that provides guidance functionality on rockets, airplanes, satellites, and scientific probes does not ever crash, as software failures have caused the loss of several vehicles in the past.[45] More advanced versions of these techniques may eventually lead to strong invariants being much more commonly and less expensively used in a wider range of applications.

Transparency advocates often claim that by reviewing a program's disclosed source code, an analyst will be able to determine how a program behaves.[46] Indeed, the very idea that transparency allows outsiders to understand how a system functions is predicated on the usefulness of static analysis. But this claim is belied by the extraordinary difficulty of identifying even genuinely malicious code ("malware"), a task which has spawned a multibillion-dollar

---

42 *See* Hoare, *supra* note 25, at 576-80.

43 *See generally* Vijay D'Silva et al., *A Survey of Automated Techniques for Formal Software Verification*, 27 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN INTEGRATED CIRCUITS & SYSTEMS 1165, 1165 (2008) ("Formal verification tools can provide a guarantee that a design is free of specific flaws. This paper surveys algorithms that perform automatic static analysis of software to detect programming errors or prove their absence.").

44 Since Heartbleed was caused by improper access to the program's main memory, *see* Felten & Kroll, *supra* note 34, computer scientists refer to the property that a program has no such improper access as "memory safety." For a discussion of the formal meaning of software safety, see PIERCE, *supra* note 24, at 95-80. For an approachable description of possible memory safety issues in software, see Erik Poll, Lecture Notes on Language-Based Security ch. 3, https://www.cs.ru.nl/E.Poll/papers/language_based_security.pdf [https://perma.cc/YJ23-6PG6]. Several modern programming languages are memory safe, including some, such as Java, that are widely used in industrial software development. However, while any system could be written in a memory safe language, developers often choose memory unsafe languages for performance and other reasons.

45 Both the Ariane 5 and Mars Polar Lander crashed due to software failures. *See* J.L. LIONS, CHAIRMAN, ARIANE 501 INQUIRY BD., ARIANE 5: FLIGHT 501 FAILURE (1996); James Gleick, *Little Bug, Big Bang*, N.Y. TIMES MAG. (Dec. 1, 1996), http://www.nytimes.com/1996/12/01/magazine/little-bug-big-bang.html [https://perma.cc/D4JE-V7K2]; *see also* ARDEN ALBEE ET AL., JPL SPECIAL REVIEW BD., REPORT ON THE LOSS OF THE MARS POLAR LANDER AND DEEP SPACE 2 MISSIONS (2000), http://spaceflight.nasa.gov/spacenews/releases/2000/mpl/mpl_report_1.pdf [https://perma.cc/RE9Z-PX6L]. Similarly, a software configuration error caused the crash of an Airbus A400M military transport. Sean Gallagher, *Airbus Confirms Software Configuration Error Caused Plane Crash*, ARS TECHNICA (June 1, 2015), http://arstechnica.com/information-technology/2015/06/airbus-confirms-software-configuration-error-caused-plane-crash [https://perma.cc/X9A4-X9CH].

46 *See* FRANK PASQUALE, THE BLACK BOX SOCIETY: THE SECRET ALGORITHMS THAT CONTROL MONEY AND INFORMATION 40 (2015).

industry based largely on the careful review of code samples collected across the internet.[47] Of course, under some circumstances, transparency can also use dynamic methods such as emulating disclosed code on disclosed input data. We discuss transparency further in Part II.

2. Dynamic Testing: Examining a Program's Actual Behavior

By running a program, dynamic testing can provide insights not available through static source code review. But again, there are limits. While static methods may fail to reveal what a program will do, dynamic methods are limited by the finite number of inputs that can be tested or outputs that can be observed. This is important because decision policies tend to have many more possible inputs than a dynamic analysis can observe or test.[48] Dynamic methods, including structured auditing of possible inputs, can explore only a small subset of those potential inputs.[49] Therefore, no amount of dynamic testing can make an observer certain that he or she knows what the computer would do in some *other* situation that has yet to be tested.[50]

Dynamic testing can be divided into "black-box testing," which considers only the inputs and outputs of a system or component, and "white-box

---

47 Malware analysis can also be dynamic. A common approach is to run the code under examination inside an emulator and then examine whether or not it attempts to modify security-restricted portions of the system's configuration. For an overview, see Manuel Egele et al., *A Survey on Automated Dynamic Malware-Analysis Techniques and Tools*, 44 ACM COMPUTING SURVEYS 6 (2012).

48 Computer scientists call this problem "Combinatorial Explosion." It is a fundamental problem in computing affecting all but the very simplest programs. Edward Tsang, *Combinational Explosion*, U. ESSEX (May 12, 2005), http://cswww.essex.ac.uk/CSP/ComputationalFinanceTeaching/CombinatorialExplosion.html [https://perma.cc/R7KE-4BJD].

49 Even auditing techniques that involve significant automation may not be able to cover the full range of possible input data if that range cannot be limited in advance to a small enough size to be searched effectively. For programmers testing their own software, achieving complete coverage of a program's behavior by testing alone is considered impossible. Indeed, if testing for the correct behavior of a program were possible at a modest cost, then there would be no bugs in modern software. For a formal version of this argument, see H.G. Rice, *Classes of Recursively Enumerable Sets and Their Decision Problems*, 74 TRANSACTIONS AM. MATHEMATICAL SOC'Y 358 (1953).

50 Computer security experts often worry about so-called "back doors," which are unnoticed modifications to software that cause it to behave in unexpected, malicious ways when presented with certain special inputs known only to an attacker. There are even annual contests in which the organizers "propose a challenge to coders to solve a simple data processing problem, but with covert malicious behavior. Examples include miscounting votes, shaving money from financial transactions, or leaking information to an eavesdropper. The main goal, however, is to write source code that easily passes visual inspection by other programmers." THE UNDERHANDED C CONTEST, http://www.underhanded-c.org/_page_id_2.html [https://perma.cc/82N4-FBDP]. Back doors have been discovered sitting undetected for many years in commercial, security-focused infrastructure products subject to significant expert review, including the Juniper NetScreen line of devices. *See* Matthew Green, *On the Juniper Backdoor*, FEW THOUGHTS ON CRYPTOGRAPHIC ENGINEERING (Dec. 22, 2015), http://blog.cryptographyengineering.com/2015/12/22/on-juniper-backdoor [https://perma.cc/M7S8-SCM4] (describing the unauthorized code that created a security vulnerability in the Juniper devices).

testing," in which the structure of the system's internals is used to design test cases. Intuitively, white-box evaluation is more powerful, since any test that can be performed in a black-box setting can also be performed in a white-box setting, but white-box evaluation can suggest more robust test cases by showing an analyst when multiple tests will yield the same behavior or what inputs are likely to trigger differences in the output.[51] White-box analysis also helps the developers and operators of a system determine how to monitor its operation so that deviations from expected behavior (e.g., unforeseen bugs, security compromise, abuse, and other unexpected behavior) can be detected and remedied.[52]

Even structured "audits" of software systems, in which systems are provided with related inputs and analyzed for differential behavior, cannot provide complete coverage of a program's behavior for the same reason: this methodology explains little about what happens to inputs which have not been tested, even those that differ very slightly.[53] Additionally, auditing that treats a system as a black box tells an analyst very little about *why* differential behavior was observed. A computer program could treat two inputs very differently because it has been explicitly designed to use special case logic for one or both, because those inputs naturally fall into different decision categories, or because the decision rule in use is very sensitive to small changes in its input.

One extremely straightforward and very commonly used form of dynamic program review comes from the practice of *logging*, or recording certain program actions in a file either immediately before or immediately after they have taken place.[54] Analysis of log messages is among the easiest and is perhaps the most common type of functional review performed on most software programs. However, analyzing program logs requires that programs be written to log when they perform events which might be interesting for analysis (and that they log enough information about those events to actually perform the analysis in question).[55] And because logs are just like other files on a computer, they can easily be modified and rewritten to contain a sequence of events that bears no relation to what a system's software actually

51 *See* MCGRAW, *supra* note 32, at 106, 109 (describing both the benefits and limitations of static code analysis).

52 *See* SLAWEK LIGUS, EFFECTIVE MONITORING AND ALERTING 1-2 (2012) (describing how to perform monitoring effectively, as opposed to verifying a system's behavior through testing alone).

53 *See infra* note 58 and accompanying text.

54 Logging is now sufficiently common to be a basic feature of most programming languages. For an overview of early uses, see Ronald E. Rice & Christine L. Borgman, *The Use of Computer-Monitored Data in Information Science and Communication Research*, 34 J. AM. SOC'Y INFO. SCI. 247 (1983).

55 *See, e.g.*, Bernard J. Jansen, *Search Log Analysis: What It Is, What's Been Done, How to Do It*, 28 LIBR. & INFO. SCI. RES. 407 (2006).

did. Because of this, *audit logs* meant to record sensitive actions requiring reliable review are generally access controlled or sent to special restricted remote systems dedicated to receiving logging data.[56]

3. The Fundamental Limit of Testing: Noncomputability

Testing of any kind is, however, a fundamentally limited approach to determining whether any fact about a computer system is true or untrue. There are some surprising limitations to the ability to evaluate code statically or dynamically. Counterintuitively, the power of computers is generally limited by a concept that computer scientists call *noncomputability*.[57] In short, certain types of problems cannot be solved by any computer program in any finite amount of time. There are many examples of noncomputable problems, but the most famous is Alan Turing's "Halting Problem," which asks whether a given program will finish running ("halt") and return an answer on a given input or will run forever on that input. No algorithm can solve this problem for every program and every input.[58] As a corollary, no testing regime can establish any property for all possible programs, since no regime can even determine whether all programs will actually terminate.[59] A related theorem, proposed by Rice, strongly limits the theoretical effectiveness of testing, saying that for any nontrivial property of a program's behavior, no algorithm can always establish whether a program under analysis has that property.[60] Any such algorithm must get some cases wrong even if the algorithm can do both static and dynamic analyses of the program.[61] However, testing can be very useful in establishing certain specific invariants on restricted classes of programs, and can be made much more useful when programs are designed

---

56 A common feature of security breaches of computer systems is that attackers will rewrite logs to prevent investigation into how the attack was carried out or who did it. *See* ERIC COLE ET AL., NETWORK SECURITY BIBLE 198 (2005) (noting that the "early stages of an attack often deal with deleting and disabling logging"). Modifying logs in this way can even allow attackers to avoid losing access to a compromised system once the compromise has been detected, since it obscures what steps must be taken to remediate the intrusion. *See generally id.* (describing how security breaches happen, how they are investigated, and how attackers try to cover the traces of their activity).

57 *See* 2 MICHAEL SIPSER, INTRODUCTION TO THE THEORY OF COMPUTATION 201 (2006) (noting the existence of "computationally unsolvable problems").

58 *See* A.M. Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem*, 42 PROC. LONDON MATHEMATICAL SOC'Y 230, 259-63 (1937) (proving that the Hilbert Entscheidungsproblem has no solution); *see also* A.M. Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction*, 43 PROC. LONDON MATHEMATICAL SOC'Y 544, 544-46 (1937) (providing a correction to the original proof).

59 To see why this is so, imagine writing a new program which halts if it decides that the program it is testing has a certain property, and which runs forever otherwise. For a more detailed version of this argument, see SIPSER, *supra* note 57, at 219, 241, 243 (discussing Rice's Theorem).

60 Rice, *supra* note 49.

61 *Id.*

to facilitate the use of testing to establish those invariants. That is, while testing is not guaranteed to work in general, it can often be useful in specific cases, especially when those cases have been designed to facilitate testing.

While both static and dynamic methods are after-the-fact assessments, as they take the computer system and its design as a given, using both approaches together is often helpful. If an analyst can establish through static methods that a program behaves identically over some class of inputs,[62] the analyst can test a single input from that class and infer the program behavior for the rest of the class. However, not every computer program will be able to be fully analyzed, even with such a combination of methods.

B. *The Importance of Randomness*

Randomness is essential to the design of many computer systems, so any approach to accountability must grapple with it.[63] However, when randomness is used, it is easy to lose accountability, since by definition any outcome which a randomized process could have produced is at least facially consistent with the design of that process.[64] Accountability for randomized

62 This can be done, for example, by noting where in a program's source code it considers input values and changes its behavior. *See infra* note 67 and accompanying text.

63 In fact, there is suggestive theoretical evidence that the power of randomness may be fundamental: there are problems for which the best known randomized algorithm performs much better than the best known deterministic algorithm. For example, the well-studied "multi-armed bandit" problem in statistics has seen wide application in the field of machine learning, where randomized decisionmaking strategies are provably more efficient than nonrandomized ones. *See, e.g.*, J.C. Gittins, *Bandit Processes and Dynamic Allocation Indices*, 41 J. ROYAL STAT. SOC'Y 148, 148 (1979) (providing a formal mathematical definition of the multi-armed bandit problem); *see also* RICHARD S. SUTTON & ANDREW G. BARTO, REINFORCEMENT LEARNING 26 (1998) (providing a general overview of the usefulness of the multi-armed bandit problem in machine learning applications).

Even outside of machine learning, there are strong indications in computer science theory that certain problems can be solved efficiently only via randomized techniques. Although it is obvious that every efficient algorithm also has an efficient randomized version (which is just rewritten to take some random bits as input and ignore them), it is conjectured but not known that the converse is not true, namely that every efficient randomized algorithm also has a deterministic version that solves the same problem with comparable efficiency. For a summary of work in this area, see Leonid A. Levin, *Randomness and Nondeterminism*, 1994 PROC. INT'L CONGRESS MATHEMATICIANS 1418. Many important problems, from finding prime numbers (which is necessary for much modern cryptography), to estimating the volume of an object (which is useful in computer graphics and vision algorithms), to most machine learning, had well-understood randomized algorithms that solved them long before they had efficient deterministic solutions (many still do not have any known efficient and deterministic algorithms).

64 For example, a winning lottery ticket with the numbers "1 2 3 4 5" is just as likely to be correct as any other ticket, and yet it seems strikingly unlikely. In a similar way, it will always be necessary when randomness is involved in a process to ensure that even outcomes that are "correct" in the sense that the system could have produced them are also correct in the sense that they fulfill the goals which necessitated randomness in the first place (e.g., in a lottery, that the winning ticket numbers not be known in advance of their selection and not be influenced by the lottery operators).

processes must determine why randomness was needed and determine that the source of that randomness and its incorporation into the process under scrutiny meets those goals.

The most intuitive benefit of randomness in a decision policy is that it helps prevent strategic behavior—i.e., "gaming" of a system. When a tax examiner, for example, uses software to choose who is audited, randomization makes it impossible for a taxpayer to be sure whether or not he or she will be audited. Those who are evading taxes, in particular, face an unknown risk of detection, which can be minimized but not eliminated, and do not know whether, or when, they should prepare to be audited. Similarly, if additional security screening is applied at random to those crossing a checkpoint, or if the procedures at the checkpoint are changed at random on a day-to-day basis, a smuggler or attacker cannot be as prepared as if the procedures were fully known in advance.[65] Additionally, studies of the performance of human guards have shown that randomization in procedures reduces boredom, thereby improving vigilance.[66]

The card game of poker illustrates a second benefit: randomness can obscure secret information. A good poker player has secret information—how good her cards are—that affects how she will bet. By occasionally bluffing, she randomizes her behavior and makes it more difficult for opponents to infer the quality of her hand.

In situations where a scarce or limited resource must be apportioned to equally deserving recipients such that not all qualified applicants can receive a share, randomness can help by fairly apportioning resources to participants in a way that cannot be controlled or predicted by those in control of the resource. For example, the Diversity Visa Lottery, considered in Part II, is a case where a random lottery allocates a scarce resource—a limited number of visas to live and work in the United States. Randomness as a source of fairness requires two attributes: first, the outcome must not be controlled by the system's operator, or else the randomness serves little purpose when compared to a model where the system's operator just chooses the winners; and second, the outcome must not be predictable, or else the operator of such a system could put its favored winners into certain slots or slip them "winning

---

65 *See, e.g.*, James Pita et al., *Deployed ARMOR Protection: The Application of a Game Theoretic Model for Security at the Los Angeles International Airport*, 2008 PROC. 7TH INT'L CONF. ON AUTONOMOUS AGENTS & MULTIAGENT SYSTEMS: INDUSTRY & APPLICATIONS TRACK 125 (describing a software system that uses a game-theoretic randomized model to improve the efficiency of police and federal air marshal patrols at the Los Angeles International Airport).

66 *See, e.g.*, RICHARD I. THACKRAY ET AL., FAA CIV. AEROMEDICAL INST., PHYSIOLOGICAL, SUBJECTIVE, AND PERFORMANCE CORRELATES OF REPORTED BOREDOM AND MONOTONY WHILE PERFORMING A SIMULATED RADAR CONTROL TASK (1975) (discussing the improvement of performance through increased unpredictability in procedures).

tickets" prior to the system's operation. Further, it is important that the random choices made when the system is run be binding upon the system's operator, so that the system cannot be run many times to control the eventual output by shopping for a favorable result among many actual runs of the system. We explore precisely how to address these issues below.

Many machine learning systems use randomization as part of their normal operation. It turns out that guessing randomly and adjusting the probability of each class of output often leads to much better performance than trying to determine the absolute best decision at any point.[67]

Finally, randomization can give computers more flexibility to perform well in unexpected environments. Consider how the Roomba robot is programmed to vacuum rooms.[68] If rules of motion were hard-coded in the software controlling the robot, an unusual furniture configuration might lead to the Roomba getting stuck in a corner or under a table or repeatedly following the same path without cleaning the rest of the room. Adding in randomized motion allows it to escape these patterns and work more effectively without the need to code in all possible room configurations. By allowing for unknowns, randomized strategies can avoid worst-case outcomes with high probability, no matter how unfriendly the environment turns out to be.[69]

However, poorly designed randomization can lead to unaccountable automated decisions. If a decision depends on a randomly selected value, then any outcome consistent with some possible value of the random choice, no matter how unlikely, must be considered valid. Concretely, if a decision is

---

67 The use of randomization is found throughout artificial intelligence and machine learning. For a survey of the field, see STUART RUSSELL & PETER NORVIG, ARTIFICIAL INTELLIGENCE: A MODERN APPROACH (1995). Some models naturally depend on randomness. *See, e.g.*, KEVIN B. KORB & ANN E. NICHOLSON, BAYESIAN ARTIFICIAL INTELLIGENCE § 1.1 (2004) (describing a model of probabilistic reasoning that depends on reckoning with randomness). Other methods simply use randomness as an efficient way to explore a large space of possible strategies, in which case it is generally necessary to try to build a model many times, evaluate the differences, and report back an estimate of confidence in the system's correct construction. *See* Volodya Vovk et al., *Machine-Learning Applications of Algorithmic Randomness*, 1999 PROC. 16TH INT'L CONF. ON MACHINE LEARNING 444 (describing one approach to integrating randomness to improve a machine learning model).

68 For a description of the Roomba's movement algorithm, see Ja-Young Sung et al., *"My Roomba Is Rambo": Intimate Home Appliances*, 2007 PROC. 9TH INT'L CONF. ON UBIQUITOUS COMPUTING 145, 152, which refers to "the randomness of Roomba's movement—generated by an algorithm designed to promote Roomba's passage across all sections of the space being cleaned."

69 More concretely, one study showed that computer-generated teaching plans customized to particular students can be less effective than lesson plans without customization if the software model used to tailor lessons to individual performance is trained on large groups that do not capture individual-specific patterns. This failure of "big data" methods trained on large groups of students to properly capture the quirks of a "small data" situation (such as a classroom-sized group of students) can be avoided by adding random deviations from the model's prediction and tracking the results of these deviations. *See, e.g.*, Yun-En Liu et al., *Trading Off Scientific Knowledge and User Learning with Multi-Armed Bandits*, 2014 PROC. 7TH INT'L CONF. ON EDUC. DATA MINING 161 (observing the introduction of small changes on result prediction).

based on the outcome of a coin flip, even if the coin is biased to land heads up 99 times out of 100, a result based on a tails up flip cannot be shown to be improper, since one out of every 100 results will be derived from the value of tails. A corrupt decisionmaker could influence this supposedly random choice, picking the value of the coin consistent with its preferred outcome, or could flip many coins and then assign the value of each flip to the set of decisions he has to make (perhaps by changing the order in which he considers decisions) in such a way as to pick the outcomes he desires. Random choices generated by a computer system can also be remade by re-running a program until the outputs match a preferred outcome. Without designing the computer system to demonstrate that this is not happening, it is very hard for a decisionmaker to prove that he has not done this. The speed of automated decisionmaking only increases this risk; while physical randomization of balls in a tumbler can only produce a small number of values per hour of effort, a computer can try thousands or millions of outcomes in a matter of minutes.

Additionally, a randomized process is not easily reproduced. For example, if it depends on interaction with its environment (e.g., the operating system on which it is running, its human user, or a database with rapidly changing records), its behavior may be altered in a nondeterministic way since that environment can change between runs.[70] One unwieldy solution to this problem is to capture all of the environment in which a program runs, so that this environment can be replayed by an analyst. However, this solution does not address how to verify all of the reasons that randomness might be needed in a process.

II. DESIGNING COMPUTER SYSTEMS FOR PROCEDURAL REGULARITY

The first goal in any plan to govern automated decisionmaking should be to enable the people overseeing the process—whether they are government officials, corporate executives, or members of the public—to know *how* a computer system makes decisions (or, at the very least, that it makes decisions based on some rule, even if that rule is not fully disclosed). A baseline requirement in most contexts is *procedural regularity*: each participant will know that the same procedure was applied to her and that the procedure was not designed in a way that disadvantages her specifically.[71] This baseline

70 One specific example is a program that chooses a random value based on the time that it has been running but takes different amounts of time to run based on what other programs are running on the same physical computer system.

71 For example, a tax auditing risk assessment should not single out individuals either by name or by identifying characteristics. If a process added extra weight to filers of a particular postal code, gender, and birth month, this could be enough to single out individuals in many cases. *See, e.g.*, Paul Ohm, *Broken Promises of Privacy: Responding to the Surprising Failure of Anonymization*, 57 UCLA L. REV. 1701, 1716-27 (2010) (showing that an individual's identity may be reverse-engineered from a small number of data points).

requirement draws on the Fourteenth Amendment principle of procedural due process. Ever since a seminal nineteenth century case, the Supreme Court has articulated that procedural fairness or due process requires rules to be generally applicable and not designed for individual cases.[72] Similarly, federal statutes articulate the requirement for procedural regularity in administrative agency actions.[73] These principles are also enshrined in the Federal Rules of Civil Procedure for civil litigation.[74]

In this Part, we will demonstrate that the tools of computer science can guarantee important elements of procedural regularity when they are incorporated in the initial design of computer systems. Specifically, these tools can assure that:

- The same policy or rule was used to render each decision.
- The decision policy was fully specified (and this choice of policy was recorded reliably) before the particular decision subjects were known, reducing the ability to design the process to disadvantage a particular individual.
- Each decision is reproducible from the specified decision policy and the inputs for that decision.
- If a decision requires any randomly chosen inputs, those inputs are beyond the control of any interested party.

After describing these properties and showing how they can be implemented, we will apply them to a case study—the Diversity Visa Lottery at the State Department—where application of these tools could greatly improve the legitimacy and fairness of an automated decision procedure.

A. *Transparency and Its Limits*

A naive solution to the problem of verifying procedural regularity is to demand transparency of the source code as well as inputs and outputs for the relevant decisions; if all of these elements are public, it seems easy to determine whether procedural regularity was satisfied. Indeed, full or partial transparency can be a helpful tool for governance in many cases,[75] and transparency has often been suggested as a remedy to accountability issues

72 *See* Marchant v. Pa. R.R., 153 U.S. 380, 386 (1894) (holding that the plaintiff had due process because "her rights were measured, not by laws made to affect her individually, but by general provisions of law applicable to all those in like condition").

73 *See* Administrative Procedure Act, 5 U.S.C. §§ 551–59 (2012) (prescribing exhaustive procedural requirements for most levels of federal administrative agency action).

74 *See* FED. R. CIV. P. 1 (noting that the rules apply "in *all* civil actions and proceedings . . . to secure the just . . . determination of *every* action and proceeding" (emphasis added)).

75 *See* Danielle Keats Citron & Frank Pasquale, *The Scored Society: Due Process for Automated Predictions*, 89 WASH. L. REV. 1, 8 (2014) (arguing that "transparency of scoring systems is essential").

for computerized systems.[76] However, transparency alone is not sufficient to provide accountability in all cases.

First and foremost, it is often necessary to keep secret the elements of a decision policy, the computer systems that implement it, key inputs, or the outcome. Keeping aspects of a decision policy secret can help prevent strategic "gaming" of a system. For example, the IRS may look for signs in tax returns that are highly correlated with tax evasion based on returns previously audited.[77] But if the public knows exactly which things on a tax return are treated as telltale signs of fraud, tax cheats may adjust their behavior and the signs may lose their predictive value for the agency. Moreover, when the decision being regulated is a commercial one, such as a decision to offer credit, a business's legitimate interest in protecting proprietary information or guarding trade secrets like the underwriting formula may be incompatible with full transparency. And in many contexts, an automated decision may use as inputs, or will create as an output, sensitive or private data that should not be broadly shared to protect business interests, privacy, or the integrity of law enforcement or investigative methods. In some cases, especially with financial or medical data, disclosure may be barred or limited by statutes or regulations.[78] Finally, in many situations—such as scoring consumers for credit or insurance risk—the purpose of the automated decision process is to determine something not directly measurable, such as the risk of defaulting on credit or claiming a loss on an insurance policy. Because these values cannot be measured directly, they are computed from proxy variables such as a consumer's credit history, income, or personal attributes. Consumers who understand these actuarial processes would be rational in attempting to control the input proxy variables, which in turn could render the scoring process less useful in elucidating unmeasurable risk.[79] Secrecy discourages strategic behavior by participants in the system and prevents violations of legal restrictions on disclosure of data.

---

76 *See* 14 C.F.R. § 255.4 (2015) (requiring transparency for airline reservation system display information); Frank Pasquale, *Beyond Innovation and Competition: The Need for Qualified Transparency in Internet Intermediaries*, 104 NW. U. L. REV. 105, 160-61 (2010) (suggesting transparency in broadband networks to hold carriers accountable for potential favoritism and discrimination).

77 *See* Jeff Reeves, *IRS Red Flags: How to Avoid a Tax Audit*, USA TODAY (Mar. 15, 2015, 12:08 PM), http://www.usatoday.com/story/money/personalfinance/2014/03/15/irs-tax-audit/5864023 [https://perma.cc/BFW5-DG34] (identifying characteristics of tax returns that trigger IRS audit).

78 *See, e.g.*, 45 C.F.R. § 164.502 (2015) (restricting the disclosure of personally identifiable information collected by health care providers).

79 In particular, consumers are rational to modify proxy variables that control their perceived risk when those variables are cheaper or easier to manipulate than the gain obtained via better treatment by the decision system. Intuitively, if proxy variables are weak and easy to alter or sometimes poorly correlated with the feature being measured (e.g., standardized test scores as a measure of student learning), they are more likely to be gamed than features which are highly proximate to the value being estimated, or which are difficult or expensive to alter (e.g., annual

Second, while transparency allows for the testing strategies described earlier (i.e., static and dynamic tests including auditing), those methods are often insufficient to verify properties of software systems if these systems have not been designed with future evaluation and accountability in mind.[80]

Third, for decision processes that involve some element of randomness, even full transparency—of the system's source code, its inputs, its operating environment, and its results—does not foreclose the possibility that an outcome could be improperly fixed in an undetectable way, as described in Section I.C.[81] A classic lottery provides an excellent example. A perfectly transparent algorithm that uses a random number generator to assign a number to each participant and has the participants with the lowest numbers win will yield results that cannot be reproduced or verified because the random number generator will produce different random numbers each time. Reviewing the code alone, or even together with the data fed into it and the environment in which it was operated, does not tell us that it was actually used to produce a particular result. By design, the process produces unpredictable results that are not reproducible.

Fourth and finally, systems that change over time cannot be fully understood through transparency alone. System designers regularly change complicated automated decision processes—such as search engine ranking methodology,[82] spam filter rules,[83] intrusion detection system methods,[84] or

income as a measure of creditworthiness in a particular transaction). *See generally* CATHY O'NEIL, WEAPONS OF MATH DESTRUCTION: HOW BIG DATA INCREASES INEQUALITY AND THREATENS DEMOCRACY (2016). In economic policymaking, this is sometimes known as Goodhart's Law, popularly formulated as "[w]hen a measure becomes a target, it ceases to be a good measure"; Goodhart formulated it more formally as "[a]ny observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." C.A.E. Goodhart, *Problems of Monetary Management: The U.K. Experience*, *in* 1 PAPERS IN MONETARY ECONOMICS (1976). Hardt and his co-authors have developed adversarial methods for designing automated decision and classification systems that remain robust even in the face of gaming. *See* Moritz Hardt et al., *Strategic Classification*, PROC. 2016 ACM CONF. ON INNOVATIONS THEORETICAL COMPUTER SCI. 111 (discussing methods to strengthen classification models).

80 *See supra* Part I.

81 There are ways to incorporate randomness that can be replicated. *See infra* subsection II.C.4.

82 For a list of updates to one search engine, see *Google: Algorithm Updates*, SEARCH ENGINE LAND, http://searchengineland.com/library/google/google-algorithm-updates [https://perma.cc/XV4Z-AFF9].

83 Many spam filters work by keeping a list of bad terms, email addresses, and computers from which to block messages. The most widely used "blacklist" is produced by the organization Spamhaus. *See SBL Advisory*, SPAMHAUS, https://www.spamhaus.org/sbl [https://perma.cc/V9LN-EPK9] (describing the Spamhaus Block List Advisory, "a database of IP addresses from which Spamhaus does not recommend the acceptance of electronic mail").

84 Intrusion detection systems work in a similar way, using a set of "signatures" to identify bad network traffic from attackers. *See* Karen Kent Frederick, *Network Intrusion Detection Signatures, Part One*, SYMANTEC (Dec. 19, 2001), https://www.symantec.com/connect/articles/network-intrusion-

the algorithms that select website ads—in response to strategic behavior by participants in the system.[85] Computer systems that choose social media posts to display to users might respond to user behavior. "Online" machine learning systems can update their model for predictions after each decision, incorporating each new observation as part of their training data. Even knowing the source code and data for such systems is not enough to replicate or predict their behavior—we also must know precisely how and when they interacted or will interact with their environment. Whether updates to the system are effected by human engineers and operators (e.g., a search engine engineer modifies the software used to rank pages) or by a machine learning system (e.g., the search engine's software discovers that users more often click the second link for a certain query instead of the first, so it reverses their order), transparency alone does little to explain either why any particular decision was made or how fairly the system operates across bases of users or classes of queries. With such systems, there is the added risk that the rule disclosed is obsolete by the time it can be analyzed. Online machine learning systems update their decision rules after every query, meaning that any disclosure will be obsolete as soon as it is made.

B. *Auditing and Its Limits*

Beyond transparency, auditing is another strategy for verifying how a computer system works. An audit treats the decision process as a black box whose inputs and outputs are visible but whose inner workings are unseen.[86] The approach has a long history in offline contexts, such as testing for discrimination in retail car negotiations.[87] For retail car negotiations, the transparency of the bargaining process for the purchase of a car is insufficient to determine if different prices are offered based on race or gender.[88]

In computer science, auditing refers to "an independent evaluation of conformance of software products and processes to applicable regulations,

---

detection-signatures-part-one [https://perma.cc/Q9XY-TQCA] (discussing signatures for network intrusion detection systems).

85 *See* JURE LESKOVEC ET AL., MINING OF MASSIVE DATA SETS ch. 8 (2d ed. 2014).

86 *See generally* Christian Sandvig et al., Auditing Algorithms: Research Methods for Detecting Discrimination on Internet Platforms (May 22, 2014), http://www-personal.umich.edu/~csandvig/research/Auditing%20Algorithms%20—%20Sandvig%20—%20ICA%202014%20Data%20and%20Discrimination%20Preconference.pdf [https://perma.cc/DS5D-3JYS] (describing algorithm audits and reviewing possible audit study designs).

87 *See* Ian Ayres, *Fair Driving: Gender and Race Discrimination in Retail Car Negotiations*, 104 HARV. L. REV. 817, 818 (1991) (using auditing to determine "[w]hether the process of negotiating for a new car disadvantages women and minorities").

88 *See id.* at 827-28 (observing that women and minorities received worse prices than white males even when using the same negotiation strategy).

standards, guidelines, plans, specifications, and procedures."[89] Auditing is intended to reveal whether the appropriate procedures were followed and to uncover any tampering with a computer system's operation. For example, there is a substantial body of literature in computer science that addresses audits of electronic voting systems,[90] and security experts generally agree that proper auditing is necessary but insufficient to assure secure computer-aided voting systems.

Computer scientists, however, have shown that black-box evaluation of systems is the least powerful of a set of available methods for understanding and verifying system behavior.[91] Even for measuring demonstrable properties of software systems, such as testing whether a system functions as desired without bugs, it is much more powerful to be able to understand the design of that system and test it in smaller, simpler pieces.[92] Approaches that attempt to review system failures simply by looking at how the output responds to changes in input are limited by either an inability to attribute a cause to those changes or an inability to interpret whether or why a change is significant.[93] Instead, software developers regularly use other, more powerful evaluation techniques.[94] These include white-box testing (in which a the person doing a test can see the system's code and uses that knowledge to more effectively search for bugs) and using programming languages that automatically preclude certain types of mistakes.[95]

---

89 IEEE Computer Society, IEEE Std 1028 - IEEE Standard for Software Reviews and Audits § 8.1 (Aug. 15, 2008), http://ieeexplore.ieee.org/document/4601584 [https://perma.cc/WLD6-VPUN].

90 *See* Joseph Lorenzo Hall, Election Auditing Bibliography (Feb. 12, 2010), https://josephhall.org/eamath/bib.pdf [https://perma.cc/L397-AATD] (collecting scholarly literature discussing audits in elections).

91 Specifically, white-box testing, in which an analyst has access to the source code under test, is generally considered to be superior; even in cases where the basic testing approach does not make use of the structure of the software (e.g., so-called "fuzz testing" where a program is subjected to randomly generated input), testing benefits from some access to the structure of programs. *See supra* note 51 and accompanying text. Also, consider the difficulties encountered in one such audit study. The authors show a causal relationship between changing sensitive, protected attributes (e.g., gender) and the advertisements presented to a user (e.g., advertisements for high-paying jobs). *See* Amit Datta et al., *Automated Experiments on Ad Privacy Settings: A Tale of Opacity, Choice, and Discrimination*, 2015 PROC. PRIVACY ENHANCING TECHNOLOGIES 92, 105-06. However, the methodology is unable to identify the mechanism of this causation or even whether the results discovered will generalize beyond the data seen in the study. *Id.* at 105.

92 See *supra* notes 20–23 and accompanying text on approaches to structuring software.

93 For example, if the output of a system is an error or other failure such as a crash, it is not obvious to an analyst how to modify the output to learn much at all.

94 *See supra* Section I.A.

95 *See supra* note 24 and accompanying text.

C. *Technical Tools for Procedural Regularity*

As we demonstrated above, transparency and auditing often do not suffice for accountability. In this Section, we introduce computational methods that can provide accountability for procedural regularity even when some information is kept secret. These methods can be used alongside transparency and auditing when appropriate and apply to all computer systems.[96]

Our approach harnesses the power of computational methods and does not take the design of the computer system as a given. Instead, we explicitly advocate for systems to be designed to use methods such as the ones described here. Nor do we give up on governance when all or part of a computer system must remain secret. We rely on several advanced techniques from computer science to enable the governance of secret decision systems: software verification, cryptographic commitments, zero-knowledge proofs, and fair random choices. Counterintuitively, even when a piece of software or the data input to it is secret, these methods can guarantee that the software and inputs satisfy the requirements for procedural regularity. They can verify that the same decision policy was used for each decision, that the policy was determined and recorded before inputs were known, and that the outcomes are reproducible. Just because a given computer system or piece of software is secret does not mean that nothing about that system can be known.

1. Software Verification

Software verification is a set of techniques for proving mathematically that software has certain properties, either by analyzing existing code or by building software using specialized tools for extracting proved correct invariants. It has been a promising field for many decades, and while many benefits are realized in research prototypes today, these methods are finding increasing industrial adoption, especially in sectors where software is safety- or security-critical and in domains with strong liability regimes.[97] While the complete verification of

---

96 While the methods we propose are general, they can be inefficient for certain applications. The cost of providing a certain level of accountability must be considered as part of the design of any policy requirement. For more detail, see Kroll, *infra* note 118.

97 While software verification has been embraced by the aviation and industrial control sectors and for some financial applications (for example, the hedge fund Jane Street regularly touts its use of formal software analysis in recruiting materials sent to computer science students), it has yet to see much adoption in the critical fields of healthcare and automotive control. *See* Jean Souyris et al., *Formal Verification of Avionics Software Products*, 2009 PROC. 2ND WORLD CONGRESS ON FORMAL METHODS 532 (describing the use of software verification at Airbus); Norbert Volker & Bernd J. Kramer, *Automated Verification of Function Block-Based Industrial Control Systems*, 42 SCI. COMPUTER PROGRAMMING 101 (2002). Indeed, researchers have effected compromises of embedded healthcare devices such as pacemakers. *See, e.g.*, Daniel Halperin et al., *Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses*, 2008 IEEE SYMP. ON SECURITY &

any program is an expensive undertaking largely reserved to technologists versed in this particular area, it is important as a matter of policy to understand the options that are available so that costs and benefits can be weighed and acted upon.

Unlike static analysis, which aims to examine already-written code for bugs or deviations from its specification, or software testing, which aims to verify that software meets a specific set of functional requirements by explicitly executing the software in a particular configuration, software verification aims to prove invariants about a program mathematically, using logic to reason about a program's behavior under all conditions.[98] Verified programs come with a mathematically checkable proof demonstrating that they have certain invariants, rendering testing for those invariants unnecessary since the proof implies that such tests will always pass.[99]

There are many ways to verify software. For instance, a program can be carefully annotated using formal logic to determine its behavior in a precise manner, though this can be expensive and will not always yield the desired

---

PRIVACY 129, 141 (finding that implantable cardioverter defibrillators are "potentially susceptible to malicious attacks that violate the privacy of patient information" and "may experience malicious alteration to the integrity of information or state"). News reports also indicate that former Vice President Dick Cheney had the remote software update functionality on his pacemaker disabled so that updating the software would require surgery, ostensibly in order to prevent remote compromise of his life-critical implant. Andrea Peterson, *Yes, Terrorists Could Have Hacked Dick Cheney's Heart*, WASH. POST (Oct. 21, 2013), https://www.washingtonpost.com/news/the-switch/wp/wp/2013/10/21/yes-terrorists-could-have-hacked-dick-cheneys-heart [https://perma.cc/VY5S-6XR6].

Additionally, researchers have also demonstrated spectacular compromises of automobile control systems, including disabling brakes, controlling steering and acceleration, and completely cutting engine power. *See* Karl Koscher et al., *Experimental Security Analysis of a Modern Automobile*, 2010 IEEE SYMP. ON SECURITY & PRIVACY 447 (performing early analyses of the security of automobile computers); *see also* Stephen Checkoway et al., *Comprehensive Experimental Analyses of Automotive Attack Surfaces*, 2011 PROC. 20TH USENIX CONF. ON SECURITY 77 (same). Subsequently, researchers have demonstrated problems in other models, including luxury models with significant telematics capabilities and remote software upgrade capability, showing that active maintenance of these software systems does not completely defend against attacks. *See, e.g.*, Jonathan M. Gitlin, *Man Hacks Tesla Firmware, Finds New Model, Has Car Remotely Downgraded*, ARS TECHNICA (Mar. 8, 2016, 11:36 AM), http://arstechnica.com/cars/2016/03/man-hacks-tesla-firmware-finds-new-model-has-car-remotely-downgraded [https://perma.cc/R9C5-9RTY] (describing an incident where a Tesla car model was hacked despite frequent software updates). Problems with spontaneous acceleration in many Toyota vehicles were later traced to software issues. *See* Phil Koopman, A Case Study of Toyota Unintended Acceleration and Software Safety (Sept. 18, 2014), https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf [https://perma.cc/VP9T-VYMF] (presenting a detailed analysis of the issue). And of course, Volkswagen designed its engine control software to defeat an emissions testing regime. For a complete timeline of the Environmental Protection Agency's actions on this matter, see *Volkswagen Light Duty Diesel Vehicle Violations for Model Years 2009–2016*, EPA.GOV, https://www.epa.gov/vw [https://perma.cc/C83U-UZLG] (last updated Nov. 7, 2016).

98 *See supra* Section I.A.

99 *See supra* notes 25–27 and accompanying text.

invariants;[100] a program can be *certified* by another program that translates it to a form guaranteed to have the desired property;[101] a program can be exhaustively tested for all possible inputs to ensure that an invariant is never violated;[102] or a program can be built using tools that allow for the careful specification of invariants (and proofs of those invariants).[103] Researchers have even verified entire operating systems using these techniques.[104] Verification can be communicated to clients in a number of ways: so called *proof-carrying code* comes with a machine-checkable proof of its verified invariants, which can be checked by a user just prior to running the

100 For one of the earliest approaches to representing programs as statements in formal logic, see Hoare, *supra* note 25, at 576-80. While Hoare's techniques form the basis of many modern methods, some methods attempt to build software that is correct by virtue of its construction, rather than analyzing software that has already been written. For an overview of different approaches and their tradeoffs, see B. BÉRARD ET AL., SYSTEMS AND SOFTWARE VERIFICATION: MODEL-CHECKING TECHNIQUES AND TOOLS (2001). For a classic reference on how to include these techniques in the software engineering process, see GHEZZI ET AL., *supra* note 30.

101 These tools are known as "certifying compilers." The advantage of a certifying compiler is that one need only expend effort verifying the certifying compiler itself, not the software being compiled, in order to prove that the desired invariant holds for the compiled software. For a description of the original concept and a first implementation, see George C. Necula & Peter Lee, *The Design and Implementation of a Certifying Compiler*, 33 ACM SIGPLAN NOTICES 333 (1998). There are many examples of certified compiler systems. See, *e.g.*, Joshua A. Kroll et al., *Portable Software Fault Isolation*, 2014 PROC. IEEE 27TH COMPUTER SECURITY FOUND. SYMP. 18 (describing a certifying software fault isolation compiler built out of CompCert's certified back end).

102 This technique, known as "model checking," could also be described as a form of static analysis. Model checking aims to verify an invariant by finding a counterexample (an input to the program which makes the invariant untrue and hence not an invariant). If a counterexample can be found, the program has a demonstrable bug. If no counterexample can be found, that invariant has been verified. *See also supra* note 24 (discussing model checking).

103 Several such programming languages exist, though one of the more successful toolkits in active research is the proof assistant Coq, which allows for writing complex programs and theorems and invariants about those programs in such a way that the proved-correct programs can be "extracted" into executable code. For an introduction to Coq, see ADAM CHLIPALA, CERTIFIED PROGRAMMING WITH DEPENDENT TYPES: A PRAGMATIC INTRODUCTION TO THE COQ PROOF ASSISTANT (2013), and YVES BERTOT & PIERRE CASTÉRAN, INTERACTIVE THEOREM PROVING AND PROGRAM DEVELOPMENT: COQ'ART: THE CALCULUS OF INDUCTIVE CONSTRUCTIONS (2004), which describe advanced programming techniques. Several large and complex programs have been written in Coq, which demonstrates that it is a robust tool capable of supporting nontrivial development tasks and proofs of correctness about those tasks. Perhaps the most famous of these was the proof of the "four-color theorem," which states that any map can be drawn using only four colors such that no border on the map uses the same color for the regions on both sides of the border. Georges Gonthier, *Formal Proof—The Four-Color Theorem*, 55 NOTICES AMS 1382 (2008). Similar tools include a theorem prover for programs written in ANSI Common Lisp 2 and the interactive theorem prover Isabelle. *See* Lawrence C. Paulson, *The Foundation of a Generic Theorem Prover*, 5 J. AUTOMATED REASONING 363 (1989) (describing the design and implementation of Isabelle).

104 *See* Gerwin Klein et al., *seL4: Formal Verification of an OS Kernel*, 2009 PROC. ACM SIGOPS 22ND SYMP. ON OPERATING SYSTEMS PRINCIPLES 207 (documenting the verification of the seL4 microkernel).

program;[105] a user can recompute the analysis used to generate the proof; or the truth of the proof can be confirmed by an entity the user trusts, with cryptography used to guarantee that the version a user is running matches the version that was verified.[106]

However, just because a program has been verified or proven correct does not mean that it has been vetted at all for correctness or compliance with a policy. Verification typically constitutes a proof that the software object in use matches its specification, but this analysis says nothing about whether the specification is sufficiently detailed, correct, lawful, or socially acceptable, or constitutes good policy. Software verification is a rapidly developing field, and the costs of building fully verified software will likely drop precipitously in the coming decades, leading to wide adoption in the software industry due to the benefits of reduced security exposure and the elimination of many types of software bugs.

2. Cryptographic Commitments

A cryptographic commitment is the digital equivalent of a sealed document held by a third party or in a safe place. It is possible to compute a commitment for any digital object (e.g., a file, a document, the contents of a search engine's index at a particular time, or any string of bytes). Commitments are a kind of promise that binds the committer to a specific value for the object being committed to (i.e., the object inside the envelope) such that the object can later be revealed and anyone can verify that the commitment corresponds to that digital object.[107] In this way, as in the envelope analogy, an observer can be certain that the object was not changed since the commitment was issued and that the committer did indeed know the value of the object at the time the commitment was made (e.g., the source code to a program or the contents of a document or computer file). Importantly, secure cryptographic commitments

---

105 *See, e.g.*, George C. Necula & Peter Lee, *Safe Kernel Extensions Without Run-Time Checking*, 1996 PROC. 2ND UNENIX SYMP. ON OPERATING SYS. DESIGN & IMPLEMENTATION 229 (describing the concept of proof-carrying code and a first implementation with applications to operating system security); *see also* George C. Necula, *Proof-Carrying Code Design and Implementation.*, *in* PROOF AND SYSTEM RELIABILITY 261 (H. Schwichtenberg & R. Steibruggen eds., 2002) (giving a detailed overview of the concepts of proof-carrying code and their development over time).

106 This approach would consist of the certifying authority making a cryptographically signed statement that it had verified the proof for a binary with a certain cryptographic hash value and the distribution of a signed copy of that piece of software. For an overview of code signing systems, see *Code Signing*, CERTIFICATE AUTHORITY SECURITY COUNCIL, https://casecurity.org/wp-content/uploads/2013/10/CASC-Code-Signing.pdf [https://perma.cc/DZU8-TA36].

107 *See* Ariel Hamlin et al., *Cryptography for Big Data Security*, *in* BIG DATA: STORAGE, SHARING, AND SECURITY 267 (Fei Hu ed., 2016) (describing cryptographic commitments as a method of verification).

are also *hiding*, meaning that knowledge of the commitment (or possession of the envelope in the analogy) does not confer information about the contents. This gives rise to the sealed document analogy: once an object is "inside" the sealed envelope, an observer cannot see it nor can anyone change it. However, unlike physical envelopes, commitments can be published, transmitted, copied, and shared at very low cost and do not need to be guarded to prevent tampering. In practice, cryptographic commitments are much smaller than the digital objects they represent.[108] Because of this, commitments can be used to lock in knowledge of a secret (say, an undisclosed decision policy) at a certain time (say, by publishing it or sending it to an oversight body) without revealing the contents of the secret, while still allowing the secret to be disclosed later (e.g., in a court case under a discovery order) and guaranteeing that the secret was not changed in the interim (for example, that the decision policy was not modified from one that was explicitly discriminatory to one that was neutral).[109]

When a commitment is computed from a digital object, the commitment also yields an opening key, which can be used to verify the commitment.[110] Importantly, a commitment can only be verified using the precise digital object and opening key related to its computation; it is computationally implausible for anyone to discover either another digital object or another opening key which will allow the commitment to verify properly. In the envelope metaphor, this is tantamount to proof that neither the envelope nor the document inside the envelope was replaced clandestinely with a different envelope or document. Any digital object (e.g., a file, document, or any string of bytes) can have a *commitment* and an *opening key* such that it is (1) impossible to deduce the original object from the commitment alone; (2) possible to verify, given the opening key, that the original object corresponds to the

108 *See id.* at 267 (noting that commitments can be smaller than the statements to which they relate). A typical commitment will be 128 or 256 bytes, regardless of the size of the committed object. *See* INFO. TECH. LAB., NAT'L INST. OF STANDARDS & TECH., FIPS PUB 180-4, SECURE HASH STANDARD (2015) (describing the hash algorithms accepted for government computer applications, which provide widely used standards in industry).

109 As a curiosity, we remark that the popular board game *Diplomacy* is essentially based on physical world commitments: each player negotiates a set of moves for the next round of the game, but then these moves are written on paper and passed secretly to a game master who stores them in an envelope. Once all players have entered their moves, the moves are revealed and taken simultaneously. This commitment mechanism allows players to simulate simultaneous moves without any risk that a player will fall behind or change their moves in a particular round in response to their perception of what another player is doing in that round. However, the commitment mechanism alone does not prevent players from entering incorrect or impossible moves, writing nonsense on their paper instead of moves, or simply refusing to enter a move at all (the game master, however, enforces that all moves placed into the envelope are correct and all players must trust her to do this to ensure that the game is not spoiled). Below, in the section on zero-knowledge proofs, we describe how techniques from computer science can address the role of the game master purely through computation without the need for an entity trusted by all players of the game.

110 *See* ODED GOLDREICH, FOUNDATIONS OF CRYPTOGRAPHY – A PRIMER 19 (2005).

commitment; and (3) impossible to generate a fake object and fake opening key such that using the (real) commitment and the fake opening key will reveal the fake object.

Cryptographic commitments have useful implications for procedural regularity in automated decisions. They can be used to ensure that the same decision policy was used for each of many decisions. They can ensure that rules implemented in software were fully determined at a specific moment in time. This means a government agency or other organization can commit to the assertions that (1) the particular decision policy was used and (2) the particular data were used as input to the decision policy (or that a particular outcome from the policy was computed from the input data). The agency can prove the assertions by taking its secret source code, the private input data, and the private computed decision outcome and computing a commitment and opening key (or a separate commitment and opening key for each policy version, input, or decision). The company or agency making an automated decision would then publish the commitment or commitments publicly and in a way that establishes a reliable publication date, perhaps in a venue such as a newspaper or the Federal Register. Later, the agency could prove that it had the source code, input data, or computed results at the time of commitment by revealing the source code and the opening key to an oversight body such as a court. This technique assures that the software implementing the decision policy was determined and recorded prior to the publication of the commitment, which can be useful in demonstrating that neither the software nor the decision policy were influenced by later information or events.

By themselves, however, cryptographic commitments do not prevent the committer from lying and generating a fake commitment that it cannot open at all or from destroying (or refusing to disclose) the information that allows a valid commitment to be opened. In either case, when the time comes to reveal the contents of the commitment, it will be demonstrable that the committer has misbehaved. However, an observer does not know the nature of the misbehavior. The committer may not have a correct opening key (analogous to having sealed an unintelligible or irrelevant document in a physical envelope) or may want to lie about what was in the original file (analogous to discovering that the contents of the envelope may be embarrassing under scrutiny of oversight). In either case, an oversight authority might punish the committer for lying and assume the worst about the contents of the missing file.[111] However, it would be preferable to be able to avoid this scenario altogether, which we can do with another tool known as zero-knowledge proofs.

---

111 A parallel to this assumption is a spoliation inference, which sanctions a party who withholds, tampers, or destroys evidence by assuming that the missing or changed evidence was unfavorable to the spoliator. *See* Fed. R. Civ. P. 37(e)(2)(A) (providing that if electronically stored

3. Zero-Knowledge Proofs

A zero-knowledge proof is a cryptographic tool that allows a decisionmaker, as part of a cryptographic commitment, to prove that the decision policy that was actually used (or the particular decision reached in a certain case) has a certain property, but without having to reveal either *how* that property is known or what the decision policy actually is.[112]

For example, consider how money flows in an escrow transaction. Traditionally, an escrow agent holds payment until certain conditions are met. Once they are, the agent attests to this fact and disburses the money according to a predetermined schedule. Zero-knowledge proofs can allow escrow without a trusted agent. Suppose that an independent sales contractor wishes to certify that she has remitted appropriate taxes from her sales to be paid by a counterparty, but without revealing precisely how much she was able to sell an item for. Using a zero-knowledge proof, she can demonstrate that sufficient taxes were paid without disclosing her sales prices or earnings to a third party.

Another classic example used in teaching cryptography posits that two millionaires are out to lunch and they agree that the richer of them should pay the bill. However, neither is willing to disclose the amount of her wealth to the other. A zero-knowledge proof allows them both to learn who is wealthier (and thus who should pay the restaurant tab) without revealing how much either is worth.

A zero-knowledge proof works with cryptographic commitments to verify procedural regularity in the following manner. If a decisionmaker makes a trio of commitments, A, B, and C, where A is a commitment to the decision policy, B is a commitment to the inputs that were used in a particular case, and C is a commitment to the decision actually reached in that case, then zero-knowledge proofs let the public verify that A, B, and C really do correspond to each other. In other words, the decisionmaker can prove that, when the committed policy A is applied to the committed input data B, the result is the committed outcome C. This allows decisionmakers to build audit logs, which can be verified by the public to confirm that the decisionmaker applied the appropriate policy to the correct input in order to reach the stated outcome, all without revealing the decision policy itself and without revealing private data that might be included in the input or outcome.

Later, if the outcome is challenged, a court or other oversight body can compel the decisionmaker to reveal the actual policy and input used and can verify that it matches the published commitment, effectively providing digital

information is lost because a party, intending to deprive the other party of the information, failed to take reasonable steps to preserve it, the court may "presume that the lost information was unfavorable to the party").

112 *See* GOLDREICH, *supra* note 110, at 16.

evidence that the decisionmaker was honest about its announced decision. By using a commitment to the same policy in decisions for multiple decision subjects, a decisionmaker can demonstrate that it is applying a consistent policy across those subjects. Such zero-knowledge proofs can be enhanced to test parts of the decision policy, either by exhibiting properties of the input–output relation (e.g., that a credit score would have been the same if the subject's gender were reversed) or properties of the policy itself (e.g., that the policy only uses certain inputs for certain purposes).

4. Fair Random Choices

Where random choices are part of a decisionmaking process, the fairness of the randomness used in those computer systems should be verifiable. Poorly designed randomization can lead to unaccountable automated decisions. The decisionmaker could influence the supposedly random choices or could generate many sets of random values and then pick the set that gives its preferred outcome. Additionally, a randomized process is not easily reproduced. For example, if it depends on interaction with its environment (e.g., the operating system on which it is running or its human user), its behavior may be altered in a nondeterministic way since that environment can change between runs.[113]

Automated decision processes must therefore be designed from the beginning to allow for oversight of the decisionmaker and to avoid problems with unpredictable behavior. To solve this problem, a decisionmaker could demonstrate that any unpredictable behavior or random choices in the software does not affect the eventual output; for example, a program designed to find the top of a hill (i.e., optimize some objective) can start at any randomly chosen point and take any arbitrary path upwards and will still ultimately return the same maximum value.[114]

More often, the random choices made by an automated decision process will affect the results. In these cases, the software implementing the decision

---

113 One example is a program that chooses a random value based on the time that it has been running but takes different amounts of time to run based on other programs that are running on the same physical computer system.

114 In general, this approach will only find the top of *some* crest, which may or may not be the highest point on a hill (for instance, if a mountain has two peaks, one much higher than the other). Randomness helps fix this problem, however, since an algorithm can start climbing the hill at many different randomly chosen points and verify that they all reach the same highest point. Additionally, for many important problems, one can prove that only a certain limited number of optimal (i.e., highest or lowest) values exist. That is, if an analyst knows that the hill has only one peak, then which path a program takes to the top is irrelevant. For a description of the gradient descent approach to optimization and other approaches, see RICHARD O. DUDA ET AL., PATTERN CLASSIFICATION 224-27 (2d ed. 2001).

can always be redesigned to replace the set of random choices made by the software with a small, recorded random input (a *seed* value) from which any necessary random values can be computed in a deterministic, *pseudorandom* way. In this way, the decisionmaking process can be replayed so long as the seed is known and the randomness of the input is completely captured by the randomness of the seed. Using this technique, a decisionmaker would not have to generate a new random choice each time a random value is needed by a piece of software (such choices can be made by a cryptographic algorithm that uses the seed to yield reproducible values), nor know in advance how many random choices must be made. This technique allows software that makes random choices, such as a lottery, to be made fully reproducible and reviewable. Unlike capturing the entire environment, as was discussed above, this technique reduces the relevant portion of the environment to a very small and manageable value (the seed) while preserving the benefits of using randomness in the system.

If this technique is used, we also must prevent the decisionmaker from tampering with the seed value, as it fully determines all random data accessed by the program implementing the decision policy. Several methods can aid in ensuring the fair choice of seed values. A public procedure can be used to select a random value: for example, rolling dice or picking ping pong balls from the sort of device used by state lotteries.[115] Alternatively, the seed value could be provided by a trusted third party, such as the random "beacon" operated by the U.S. National Institute of Standards and Technology (NIST).[116] In addition, it is possible for a set of mutually distrustful parties

115 Currently known strategies for generating public random values ("randomness beacons") all have advantages as well as disadvantages—dice could be weighted; ping pong balls could be put in the freezer and the cold ones picked out of the machine. The National Institute of Standards and Technology runs a randomness beacon that has come under scrutiny because of distrust of the National Security Agency. To minimize these types of issues, the algorithm designer should pick the source of randomness most likely to be trusted by participants, which may vary. The algorithm designer could choose to collect many sources of random choices and mix them together to maximize the number of participants who will trust the randomness of the chosen seed. However, even physical sources of randomness that have not been tampered with have failed to be accountable for their goals in unexpected ways; for instance, the 1969 lottery for selecting draftees by birthday was later shown to be biased, with a disproportionate number of selectees coming from months early in the year. For a detailed overview of the problem and its causes, see Joan R. Rosenblatt & James J. Filliben, *Randomization and the Draft Lottery*, SCIENCE, Jan. 22, 1971, at 306.

116 Computer science refers to a trusted third-party source of randomness as a "beacon." The best known beacon is operated by the NIST, which publishes new random data every few minutes, ostensibly based on the measurement of quantum mechanical randomness via a device maintained in a NIST lab. *NIST Randomness Beacon*, NAT'L INST. STANDARDS & TECH., http://www.nist.gov/itl/csd/ct/nist_beacon.cfm [https://perma.cc/UNT3-6N6P] (last updated Sept. 21, 2016). Recent revelations about NIST's role in allowing the U.S. National Security Agency to undermine the security of random number generation techniques standardized by NIST have led to some distrust of the NIST beacon, although it may be trustworthy in some applications. NIST standardized the

(possibly including decision subjects themselves) to engage in an interaction that produces a value that is unpredictable so long as at least one participant provided random input.[117] Perhaps the best option is to mix together randomness (sometimes called entropy) from many different sources. The simplest form of this practice would involve a decision subject entering a short random number as part of the input for their decision (e.g., on an application form). Then, the decisionmaker would generate a seed value for each decision by combining this known-to-the-subject, personal random value with (1) a pre-chosen random value to which the decisionmaker publicly committed to using far in advance of seeing the personal random value, and (2) a unique identifier for the particular decision or decision subject that is difficult to change (e.g., the social security number of the participant). In order to foster maximum confidence that random choices are not improperly influenced, the decisionmaker should derive them using a combination of (1) a random value from a trusted third-party; (2) a random value chosen by the decisionmaker and possibly kept secret; (3) a participant- or decision-specific identifier that cannot be changed or controlled by the decisionmaker, such as a social security number, identification number, or other immutable piece of the subject's name or data; and (4) a value chosen by the decisionmaker. Since these values are either outside of the decisionmaker's control or are known, fixed, and subsequently verifiable before the inputs to a decision are known, using these methods gives assurance that the decisionmaker is not skewing the results by controlling the selection of random values.[118]

Duel EC Deterministic Random Bit Generator (DUAL-EC) in SP 800-90A in 2007. At that time, cryptographers already knew the standard could accommodate a "backdoor," or secret vulnerability. *See* Dan Shumow & Niels Ferguson, On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng (2007), http://rump2007.cr.yp.to/15-shumow.pdf [https://perma.cc/VC7G-G23G]. Later, it was discovered that the NSA had very likely made use of this mechanism to create a backdoor in the standard itself. *See* Daniel J. Bernstein et al., *Dual EC: A Standardized Back Door*, *in* THE NEW CODEBREAKERS 256 (2016). Other beacon implementations have been proposed, including beacons based on "cryptocurrencies" such as Bitcoin. *See, e.g.*, Joseph Bonneau et al., On Bitcoin as a Public Randomness Source, https://eprint.iacr.org/2015/1015.pdf [https://perma.cc/XQ38-FJ3H] (outlining a specific alternate proposal involving the use of Bitcoin as a source of publicly verifiable randomness).

117 Computer science has methods to simulate a trusted third party making a random choice. These methods require the cooperation of many mutually distrustful parties, such that as long as any one party chooses randomly, the overall choice is random. By selecting many participants in this process, one can maximize the number of people who will believe that the chosen value is in fact beyond undue influence. For an easy-to-follow introduction to these methods, see Manuel Blum, *Coin Flipping by Telephone: A Protocol for Solving Impossible Problems*, 15 ACM SIGACT NEWS 23 (1983).

118 When the fairness of random choices is key to the accountability of a decision process, great care must be taken in determining the source of random seed values, as many very subtle accountability problems are possible. For example, by changing the order in which decisions are taken, the decisionmaker can effectively "shop" for desirable random values by computing future deterministic pseudorandom values and picking the order of decisions based on its preference for which decisions receive which random choices. To prevent this, it may also be necessary to require

Where random choices are part of a decisionmaking process, the fairness of the randomness (i.e., the consistency with the goal for which randomness was deployed in a particular system) used in those decisions should be verifiable. This can be achieved by relying on a small random seed and verifying its source. Once a random seed has been chosen in a satisfactory manner, it is still necessary to verify that the seed was in fact used in later decisions.[119] This can be accomplished by the techniques we describe.

D. *Applying Technical Tools Generally*

Our general strategy in designing systems accountable for their procedural regularity is to require systems to create cryptographic commitments as digital evidence of their actions. Systems can be designed to publish commitments describing what they will do (i.e., a commitment to the decision policy enforced by the system, as represented by source code) before they are fielded and commitments describing what they actually did (i.e., a commitment to the inputs[120] and outputs of any particular decision) after they are fielded. Zero-knowledge proofs can be used to ensure that these commitments actually correspond to the actions taken by a system.[121] Indeed, it is possible to use zero-knowledge proofs to verify, for each decision, that the committed-to decision policy applied to the committed-to inputs yields the committed-to outputs.[122] These zero-knowledge proofs could either be made public or provided to the system's decision subjects along with their results.

By disclosing commitments instead of source code or inputs and outputs, system operators can fully explain what their systems do without actually disclosing how those systems work up front. Later, if it becomes necessary to

---

that the decisionmaker take decisions in a particular order or that the decisionmaker commit to the order in which it will take decisions in advance of the seed being chosen. For a detailed description of the problems with randomness "shopping" and post-selection by a decision authority, see Joshua Alexander Kroll, Accountable Algorithms (Sept. 2015) (unpublished Ph.D. dissertation, Princeton University) (on file with author).

119 For example, several state lotteries have been defrauded by insiders who were able to control what random values the lottery system used to decide winners. Specifically, an employee of the Multi-State Lottery Association (MUSL) was convicted of installing software on the system that controlled the random drawing and using the information gleaned by the software to purchase winning tickets for the association's "Hot Lotto" game. *See* Grant Rodgers, *Hot Lotto Rigger Sentenced to 10 Years*, DES MOINES REG. (Sept. 9, 2015, 7:12 PM), http://www.desmoinesregister.com/story/news/crime-and-courts/2015/09/09/convicted-hot-lotto-rigger-sentenced-10-years/71924916 [https://perma.cc/U26A-8VMD] (describing the Iowa lottery fraud sentencing).

120 Note that, for these commitments to function, systems must also be designed to be fully reproducible, capturing all interactions with their environments as explicit inputs that can then be contained in published commitments. The use of seed values for randomization, discussed above in subsection II.C.4, offers one example of ensuring reproducibility.

121 The approach here was introduced in Kroll, *supra* note 118.

122 *Id.*

review the actions or decision policy of a system during a court case or regulatory action, system operators can disclose the contents of their commitments (that is, source code, inputs, and outputs), possibly under a protective regime. If it is possible to disclose these values publicly, then system operators may also choose (or be required) to do so. However, whether these data are disclosed or not, the published commitments and zero-knowledge proofs allow overseers and the public at large to verify that the decisions of some authority actually correspond to a specific predetermined policy rather than the arbitrary whim of a decisionmaker. Further, by observing that all decisions arise from the same policy, anyone reviewing these commitments can be certain that the policy was used for all decisions simply by checking that the commitments to the decision policy are consistent across decisions.

By requiring commitments to be published far in advance of any decision, it is possible to ensure that the particulars of a decision policy were chosen independently of the factors in the decisions it would render. For example, a decision policy that selects which individuals will receive a tax audit should be based on the risk of tax evasion, which in turn can be inferred by properties of the tax return itself. However, a corrupt tax authority could pick out individuals for audit and guess the particulars of their tax return data, then tailor the audit decision policy accordingly. Further, if a policy must be approved in advance by some oversight or certification body, the policy would need to be decided on and implemented in software far enough in advance to admit certification or review. Finally, if such certification does take place, subjects of the policy's decisions (or overseers of those decisions) can be certain after the fact that the policy which was certified is the policy which was actually used in practice.[123]

To the extent that the invariants of interest in a computer system are simple enough to compute, their truth can be attested by the same zero-knowledge proofs that attest to the relationship between the code, the inputs, and the outputs. Because powerful, modern, zero-knowledge techniques can be applied to any code, they can also be applied to code that performs the analysis of these

123 Electronic voting systems have suffered from such problems in practice. In many jurisdictions, voting system software must be certified before it can be used in polling places. Systems are tested by the Election Assistance Commission (EAC), an independent commission created by the 2002 Help America Vote Act. *See Testing and Certification Program*, U.S. ELECTION ASSISTANCE COMMISSION, http://www.eac.gov/testing_and_certification [https://perma.cc/8DFX-LTYD] (detailing the EAC's testing and certification regime). However, in many cases, updated, uncertified software has been used in place of certified versions because of pressure to include updated functionality or bug fixes. *See, e.g.*, Fitrakis v. Husted, No. 2:12-cv-1015, 2012 WL 5411381 (S.D. Ohio Nov. 6, 2012) (involving a suit arising out of updates to voting systems immediately prior to the 2012 presidential election in Ohio).

invariants, and the execution of that code can be considered as part of the operation of the system.[124]

Simply publishing commitments to the inputs and outputs of a system rather than making them transparent will not solve all of the issues with transparency brought up in Part I. However, it can address the need for legitimate secrecy of the system, its inputs, or its outputs. Because it is possible, using these methods, to verify that a particular input and a particular decision policy correspond to a particular output, it is not strictly necessary to see these values in order to investigate the system's procedural regularity.

We describe how certification of procedural regularity can be done for randomized software, such as software implementing a lottery, in greater detail in Section II.E below. Later, in Part III, we explain how these tools can extend to certify other, more complicated invariant properties of interest, enabling proof that a system comports with substantive goals or principles beyond simple procedural regularity.

E. *Applying Technical Tools to Reform the Diversity Visa Lottery*

Armed with these tools, we can turn to the question of how to ensure the procedural regularity of automated decisionmaking. To illustrate how designing a computer system can make it more accountable, we will apply the methods described above to a case study: the Diversity Visa Lottery (DVL) operated by the U.S. Department of State.

1. Current DVL Procedure

The DVL is run annually by the State Department to grant U.S. permanent resident visas ("green cards") to 50,000 immigrants from around the world. The process, prescribed by 8 U.S.C. § 1153(c), is intended to increase the national and regional diversity of immigrants to the United States by granting visas to a sample of people from countries otherwise underrepresented in the immigrant population.

The annual DVL process operates as follows.[125] Would-be immigrants apply to be entered in the lottery, applicants are grouped according to their country of birth, and countries are assigned to one of six regional groups. Within each group, applicants are put into a rank-ordered list in a random order (the lottery step). The Attorney General then calculates the number of

---

124 For example, suppose that we wish to demonstrate that a decision would be the same if the subject's gender were reversed. The software implementing the decision could simply compute the decision with a different gender for each subject and confirm that the same result is reached in each case.

125 *See generally* Immigration and Nationality Act § 203(c), 8 U.S.C. § 1153(c) (2012); U.S. DEP'T OF STATE, FOREIGN AFFAIRS MANUAL ch. 9, § 502.6.

applicants to accept from each region using a formula based on the number of immigrants to the United States in recent years from each other region. The calculated number of applicants is selected from the top of each group's rank-ordered list. These "winners" are screened for eligibility to enter the United States, and they receive visas if they are eligible. In some years, additional winners are selected so that all statutorily available visas can eventually be awarded, even if some applicants fail the screening process, drop out, or fail to proceed with their visa application.[126]

Questions have been raised about the correctness and accountability of this process. Would-be immigrants sometimes question whether the process is truly random or, as some suspect, is manipulated in favor of individuals or groups favored by the U.S. government. This suspicion, in turn, may subject DVL winners to reprisals, on the theory that winning the DVL is evidence of having collaborated secretly with U.S. agencies or interests.

There have also been undeniable failures in carrying out the DVL process. For example, the 2012 DVL initially reported incorrect results due to programming errors coupled with lax management.[127] An accountable implementation of the DVL could address both issues by demonstrating that there is no favoritism in the process and by making it easy for outsiders to check that the process was executed correctly.

2. Transparency Is Not Enough

The DVL is an automated decision system for which transparency alone cannot solve its problems. First, the software implementing the decisions appears to be written in an irreproducible way.[128] The system relies on the computer's operating system to provide random numbers; thus, attempts to replicate the program's execution at another time or on another computer will yield different random numbers and therefore a different DVL result. Notably, no amount of reading, analyzing, or testing of the software can remedy the nonreplicability of this software.

Second, the privacy interests of participants bar full transparency. People who apply to the DVL do not want their information, or even the fact that they applied, to be published. However, such publication is needed for the process to be verified through transparency and auditing. The State

126 *Visa Bulletin for June 2015*, U.S. DEP'T ST. BUREAU CONSULATE AFF., https://travel.state.gov/content/visas/en/law-and-policy/bulletin/2015/visa-bulletin-for-june-2015.html [https://perma.cc/7H7L-SJKX].

127 Memorandum from Howard W. Geisel, Deputy Inspector Gen., U.S. Dep't of State, Review of the FY 2012 Diversity Visa Program Selection Process, ISP-I-12-01 (Oct. 13, 2011), https://oig.state.gov/system/files/176330.pdf [https://perma.cc/4FWM-URYJ].

128 *Id.*

Department could try to work around this problem by assigning an opaque record ID to each applicant and then having the lottery choose record IDs rather than applicants, but lottery operators could manipulate the outcome by retroactively assigning winning record IDs to people they wanted to favor. Further, it would be difficult to verify that no extra record IDs corresponding to actual participants had been added.

3. Designing the DVL for Accountability

Instead of this inherently unverifiable approach, we propose a technical solution for building an accountable version of the DVL.[129] Using the techniques we have described, the State Department can demonstrate that it is running a fair lottery among a hidden set of participants.[130]

We can solve the nonreplicability problem by choosing a random seed, as described previously. The third-party generating the random value used to create the seed could be one or more trusted NGOs, or applicants could provide a "PIN" on their applications.

Recall that the decision policy for the DVL is fixed in statute and hence publicly known. To provide oversight, the State Department could publish in the Federal Register a commitment to its software source code (far in advance of any decisions being made) and a commitment to all the inputs (i.e., to each data element in an application for the US visa) used to create the rank-ordered list and calculate the cut-off points. The State Department would also need to provide a zero-knowledge proof showing that applying the committed-to software to the committed-to inputs produces the announced lottery results. The proof should also demonstrate that the commitment to the software published in advance of all decisions is a commitment to the same software as the one used in each individual decision. These actions would bind the State Department to its choices of software, source code, and applicant data; ensure that the commitment to the software was not a fake; and prove that the same procedure was used to render each decision. Subsequent auditing by an oversight body should establish that the source code in the commitment faithfully implements the policy specified by statute (the code should be designed to enable this).

---

129 A full technical analysis is beyond the scope of this Article.

130 Note that it is less straightforward to prove that the set of participants actually considered in the lottery matches the set of individuals who applied to be included. For example, the operator of the lottery might insert "shills," or lottery entries that do not correspond to any real applicant, and if one of these applications were to be chosen, that place could be given improperly to anyone of the Department's choosing. It is technically nontrivial to prove that no extra applications were considered; studies of end-to-end secure voting protocols provide methods to do so. *See, e.g.*, Daniel Sandler et al., *VoteBox: A Tamper-Evident, Verifiable Electronic Voting System*, 2008 PROC. 17TH CONF. ON SECURITY SYMP. 349 (enunciating the measures necessary to make electronic voting secure).

Finally, the State Department should determine an adequate method for generating a random seed to be used in the lottery step. This method should guarantee to the public that it is not possible for the State Department to choose winners by rearranging applications.[131] This could be accomplished by combining random data chosen at a public ceremony (as is done for state lotteries); alternatively, the State Department could cooperate with interested NGOs to produce a verifiable random seed with a random value selected exclusively by the State Department (and published prior to the ceremony and any lottery applications) along with something that identifies a particular lottery entry uniquely (e.g., the applicant's full application data, reduced by cryptographic hash, to a small numeric value). Depending on the implementation and application, the State Department could also include randomness selected by DVL applicants on their application, which could be harvested passively by tracking mouse movements during the application process.[132]

Once these steps are taken, each applicant can be assured that the State Department's decision on his application is fully explainable. If the applicant has questions regarding the process or a governmental overseer wants to audit it, the decisions will be replicable, and, if necessary, the secret source code and secret input data (including the random choices made in the lottery step) can be revealed and verified—by a court or auditing agency—to be the proper code and data used to render the decision.[133]

These solutions depend on both redesigning the software code (a technical solution) and adopting procedures relating to how the software program is used (a legal or policy solution). They must be deployed during the design of the decision process and cannot salvage a poorly designed system after the fact. In hindsight, it should not be surprising that the path to accountability for computational processes requires some redesign of the processes themselves. The same is true for noncomputational administrative processes, where the most accountable processes are those that are designed with accountability in mind.

---

131 Random choices in the DVL must be demonstrably random even to nonparticipants so that winners can plausibly claim that they were chosen by lottery and not because of sympathy for U.S. interests.

132 *See supra* note 117 and accompanying text.

133 In fact, just as the applicant can be convinced that his decision is explainable without seeing the secret algorithm or secret inputs, an oversight body can be convinced that particular decisions were made correctly without seeing the applicant's inputs, which might contain sensitive data like health records or tax returns. Thus, subsequent auditing is rendered more useful and more acceptable to participants, as it can determine the basis for every decision without revealing sensitive information.

III. DESIGNING ALGORITHMS TO ASSURE FIDELITY TO SUBSTANTIVE POLICY CHOICES

In Part I, we described methods that permit certification of properties of computer systems, and in Part II, we demonstrated how those methods can ensure that automated decisions are reached in accordance with agreed upon rules, a goal we called procedural regularity. In this Part, we examine how those methods could be used to certify other system properties that policymakers desire. Accountability demands not only that we certify that a policy was applied evenly across all subjects, but also that those subjects can be certain that the policy furthers other substantive goals or principles. A subject may want to know: Is the rule correctly implemented? Is it moral, legal, and ethical? Does it operate in the aggregate with fidelity to substantive policy choices?

We focus here on the goal of nondiscrimination,[134] in part because specific, additional technical tools have developed to assist with it, and in part because the use of automated decisionmaking already has raised concerns about discrimination and the ability of current legal frameworks to deal with technological change.[135] The well-established potential for unfairness in systems that use machine learning, in which the decision rule itself is not programmed by a human but rather inferred from data, has heightened these discrimination concerns. However, what makes a rule unacceptably discriminatory against some person or group is a fundamental and contested question. We do not address that question here, much less claim to resolve it with computational precision. Instead, we describe how an emerging body of computer science techniques may be used to avoid outcomes that could be considered discriminatory.

Fidelity to policy choices like nondiscrimination is a more complicated goal than procedural regularity, and the solutions that currently exist to address it are less robust. Technical tools offer ways to ameliorate these problems, but they generally require a well-defined notion of what sort of fairness they are supposed to be enforcing. In this Part, we outline a few proposed well-defined notions. We present these techniques as examples of system properties that could be certified using the techniques described in

---

134 The word "discrimination" carries a very different meaning in engineering conversations than it does in public policy. Among computer scientists, the word is a value-neutral synonym for differentiation or classification: a computer scientist might ask, for example, how well a facial recognition algorithm successfully discriminates between human faces and inanimate objects. But, for policymakers, "discrimination" is most often a term of art for invidious, unacceptable distinctions among people—distinctions that either are, or reasonably might be, morally or legally prohibited. We use the latter meaning here.

135 *See* PASQUALE, *supra* note 46, at 8-9 (describing the problem of discrimination through the use of automated decisionmaking).

Part I, but we do not necessarily advocate for any of them; ultimately, policymakers must decide whether these properties or others square with nondiscrimination goals.

In addition, the precision of computer code often brings into sharp focus the tensions within current legal frameworks for antidiscrimination. Computers favor hard and fast rules over the types of standards and balancing tests often found in our common law system and civil rights law. While these characteristics of the current legal approach suggest that doctrinal reform may be necessary to apply computerized decisionmaking in an area, we are not advocating a policy regime entirely made of bright-line rules or predetermined fairness criteria. In fact, we believe that investigations of fairness should always be in the purview of ex post review processes. Instead, we offer an overview of the problem of algorithmic discrimination, the current state of the related technical tools, and the relationship of these tools to existing legal frameworks. We describe the types of properties that can be specified in advance and certified to be in force, even if none of the properties is sufficient on its own to guarantee compliance with a policy regime. Our aim is to both elucidate the current state of the art and suggest directions for further research and action.

A. *Machine Learning, Policy Choices, and Discriminatory Effects*

We focus here on decisions developed through machine learning—on situations where a machine has been "trained" through exposure to a large quantity of data and infers a rule from the patterns it observes. Computers are especially well-suited to discover patterns in these input–output pairs that can then guide future decisionmaking. In contrast to human-made rules, these rules for decisionmaking are induced from historical examples—they are, quite literally, rules learned by example. Humans orchestrate a computerized rule-creation process, rather than imparting the rules directly. These kinds of decisions raise problems for the methods described in Part I because the system's designer does not fix the decision rule directly, and, as a result, the rule cannot directly be verified in the manner we have described. Instead, for the tools to show that such systems meet policy goals, policymakers must determine the substantive properties that the systems should have, and, if such properties exist, the tools of Parts I and II can then be used to demonstrate that techniques for certifying such properties are in use and implementers can then design the systems to allow the certification of these properties and permit the type of accountability we have proposed above.

Machine learning is an increasingly common approach to solving problems that once seemed computationally intractable due to their complexity (e.g., object recognition in a photograph). The recent movement of software systems

into a growing number of domains owes primarily to successful applications of machine learning, which is thus the primary focus of our analysis.

A significant concern about automated decisionmaking is that it may simultaneously systematize and conceal discrimination. Because it can be difficult to predict the effects of a rule in advance (especially for large, complicated rules or rules that are machine-derived from data), regulators and observers may be unable to tell that a rule has discriminatory effects. In addition, decisions made by computers may enjoy an undeserved assumption of fairness or objectivity.[136] However, the design and implementation of automated decision systems can be vulnerable to a variety of problems that can result in systematically faulty and biased determinations.[137]

These decision rules are machine-made and follow mathematically from input data, but the lessons they embody may be biased or unfair nevertheless. Below, we describe a few illustrative ways that *models*, or decision rules derived from data, generated through machine learning, may turn out to be discriminatory. We adapt a taxonomy laid out in previous work by Solon Barocas and Andrew D. Selbst[138] and make use of the "catalog of discriminatory evils" of machine learning systems laid out by Hardt[139] and Dwork et al.[140]

First, algorithms that include some type of machine learning can lead to discriminatory results if the algorithms are trained on historical examples that reflect past prejudice or implicit bias, or on data that offer a statistically distorted picture of groups comprising the overall population. Tainted training data would be a problem, for example, if a program to select among job applicants is trained on the previous hiring decisions made by humans, and those previous decisions were themselves biased.[141] Statistical distortion,

---

136 *See* Paul Schwartz, *Data Processing and Government Administration: The Failure of the American Legal Response to the Computer*, 43 HASTINGS L.J. 1321, 1342 (1992) (describing the deference that individuals give to computer results as the "seductive precision of output").

137 *See id.* at 1342-43 (noting that the computer creates "new ways to conceal ignorance and subjectivity" because people overestimate its "accuracy and applicability").

138 *See* Barocas & Selbst, *supra* note 8 (describing a taxonomy that isolates specific technical issues to create a decisionmaking model that may disparately impact protected classes).

139 Moritz A.W. Hardt, A Study of Privacy and Fairness in Sensitive Data Analysis (Nov. 2011) (unpublished Ph.D. dissertation, Princeton University) (on file with author).

140 Cynthia Dwork et al., *Fairness Through Awareness*, 2012 PROC. 3RD INNOVATIONS THEORETICAL COMPUTER SCI. CONF. 214.

141 *See* Barocas & Selbst, *supra* note 8, at 682 (citing Stella Lowry & Gordon Macpherson, *A Blot on the Profession*, 296 BRIT. MED. J. 657, 657 (1988)) (describing how a hospital developed a computer program to sort medical school students based on previous decisions that had disfavored racial minorities and women). Another example is a Google algorithm that showed ads for arrest records much more frequently when black-identifying names were searched than when white-identifying names were searched—likely because users clicked more often on arrest record ads for black-identifying names and the algorithm learns from this behavior with the purpose of maximizing click-throughs. *Id.* at 682-83 (citing Latanya Sweeney, *Discrimination in Online Ad Delivery*, COMM. ACM, May 2013, at 44, 47 (2013)).

even if free of malice, can produce similarly troubling effects: consider, for example, an algorithm that instructs police to stop and frisk pedestrians. If this algorithm has been trained on a dataset that overrepresents the incidence of crime among certain groups (because these groups have historically been the target of disproportionate enforcement), the algorithm may direct police to detain members of these groups at a disproportionately high rate (and nonmembers at a disproportionately low rate). Such was the case with the New York City Police Department's stop-and-frisk program, for which data from 2004 to 2012 showed that 83% of the stops were of black or Hispanic persons and 10% were of white persons in a resident population that was 52% black or Hispanic and 33% white.[142] Note that the overrepresentation of black and Hispanic people in this sample may lead an algorithm to associate typically black or Hispanic traits with stops that lead to crime prevention, simply because those characteristics are overrepresented in the population that was stopped.[143]

Second, machine learning models can build in discrimination through choices in how models are constructed. Of particular concern are choices about which data models should consider, a problem computer scientists call *feature selection*. Three types of choices about inputs could be of concern: using membership in a protected class directly as an input (e.g., decisions that take gender into account explicitly); considering an insufficiently rich set of factors to assess members of a protected class with the same degree of accuracy as nonmembers (e.g., in a hiring application, if fewer women have been hired previously, data about female employees might be less reliable than data about male employees); and relying on factors that happen to serve as proxies for class membership (e.g., women who leave a job to have children lower the average job tenure for all women, causing this metric to be a known proxy for gender in hiring applications). Eliminating proxies can be difficult, because proxy variables often contain other useful information that an analyst wishes the model to consider (for example, zip codes may indicate both race and differentials in local policy that is of legitimate interest to a lender). The case against using a proxy is clearer when alternative inputs could yield equally effective results with fewer disadvantages to protected class members. A problem of insufficiently rich data might be remedied in some cases by gathering more data or more features, but if discrimination is already systemic, new data will retain the discriminatory impact. While it is tempting

142 David Rudovsky & Lawrence Rosenthal, *Debate: The Constitutionality of Stop-and-Frisk in New York City*, 162 U. PA. L. REV. ONLINE 117, 120-21 (2013).

143 The underrepresentation of white people would likely cause the opposite effect, though it could be counter-balanced if, say, the police stopped a subset of white people who were significantly more likely to be engaged in criminal behavior.

to say that technical tools could allow perfect enforcement of a rule barring the use of protected attributes, this may in fact be an undesirable policy regime. As previously noted, there may be cases where allowing an algorithm to consider protected class status can actually make outcomes fairer. This may require a doctrinal shift, as, in many cases, consideration of protected status in a decision is presumptively a legal harm.

Third and finally, there is the problem of "masking": intentional discrimination disguised as one of the above-mentioned forms of unintentional discrimination. A prejudiced decisionmaker could skew the training data or pick proxies for protected classes with the intent of generating discriminatory results.[144] More pernicious masking could occur at the level of designing a machine learning model, which is a very human-driven, exploratory process.[145]

B. *Technical Tools for Nondiscrimination*

As mentioned in the previous Part, transparency and after-the-fact auditing can only go so far in preventing undesired results. Ideally, those types of ex post analyses should be used in tandem with powerful ex ante techniques during the design of the algorithm. The general strategy we proposed in Section II.D—publishing commitments and using zero-knowledge proofs to ensure that commitments correspond to the system's decisionmaking actions—can certify any property of the decision algorithm that can be checked by a second examination algorithm.[146] Such properties can be verified by making the examination algorithm public and giving a zero-knowledge proof that, if the examination algorithm were run on the secret decision algorithm, it would report that the decision algorithm has the desired property. The question then is which, if any, properties policymakers would want to build into particular decision systems.

A simple example of such a property would be the exclusion of a certain input from the decisionmaking process. A decisionmaker could show that a particular algorithm does not directly use sensitive or prohibited classes of information, such as gender, race, religion, or medical status.

The use of machine learning adds another wrinkle because decision rules evolve on the fly—they are not specified directly, but are inferred from the data. However, the absence of static, predetermined decision rules does not necessarily preclude the use of our certification strategy. Computer scientists,

144 *See* Barocas & Selbst, *supra* note 8, at 692-93 (describing ways to intentionally bias data collection in order to generate a preferred result).

145 In other words, the machine learning model would be intentionally coded to develop bias.

146 Such an algorithm might be a tool for verifying properties of software or simply a software test. *See supra* Part I (discussing software testing and software verification in greater detail).

including Hardt,[147] Dwork et al.,[148] and others, have developed techniques that formalize fairness in such a way that they can constrain the machine learning process so that learned decision rules have specific well-defined fairness properties. These methods also can be incorporated in the design of systems such that their inclusion in the decisionmaking process implies the incorporation of fairness properties that can be certified and proven.[149]

We describe three such methods below. First, decision systems can incorporate randomness to maximize the gain of learning from experience. Second, computer science offers many emerging approaches to maximize fairness, defined in a variety of ways, in machine learning systems. At a high level, all of these definitions reduce to the proposition that similarly situated people should be treated similarly, without regard to sensitive attributes. As we shall see, simple blindness to these attributes is not sufficient to guarantee even this simplified notion of fairness. Finally, related ideas from differential privacy can also be used to guarantee that protected status could not have been a substantial factor in certain decisions.

Those who use algorithmic decisionmaking today regularly make assertions about properties of these algorithms without proving them. This likely occurs because they are required by law to disclose certain facts about their decision process to regulators and consumers,[150] they simply want to generate good will, or they demonstrate better behavior than a competitor. But without proof, these assertions are just words on paper, subject to challenge by skeptical regulators and disbelief by skeptical consumers. This skepticism is not entirely unfounded: these assertions have proved false in the past. Digital evidence, such as zero-knowledge proofs, gives a direct connection between the fact being asserted and the technical mechanism of decisionmaking. This proof provides the consumer with a high assurance that the assertion proffered relates meaningfully to the facts on the ground.

1. Learning from Experience

As mentioned in Section I.B, incorporating randomness into an algorithm can give it flexibility to operate outside of the environment for which it was designed. Similarly, randomness can prevent hidden biases in the design or

147 Hardt, *supra* note 139.

148 Dwork et al., *supra* note 140.

149 We concentrate on certification and proof of a system property to an overseer, observer, or participant. However, these tools are also valuable for compliance (since proofs can certify to the implementer of a system that the system is working as intended) and for demonstration that a decisionmaker will be able to show how and why he or she used certain data after the fact in case of an audit or dispute.

150 *See, e.g.*, 12 C.F.R §§ 203.4–5 (2015) (providing requirements for the compilation, disclosure, and reporting of loan data).

deployment of an algorithm from leading to consistent discriminatory outcomes. There is a large and rich literature on how to maximally learn from previous data and how to use random choices to ensure that a model is as faithful as possible to the real world.[151]

Consider a machine learning algorithm for hiring that is trained using a biased set of initial data indicating that women are weak candidates, even though gender does not predict job performance among the full population. If the resulting model would hire mostly men, the algorithm for hiring can create a self-fulfilling prophecy in which it finds that characteristics of successful hires correlate strongly with proxies for gender. But, if the algorithm is designed to incorporate an element of randomness such that some candidates who are not predicted to do well get hired (and have their performance tracked), the validity of the initial assumptions can be tested and the accuracy and fairness of the entire system will benefit over time. By occasionally guessing about candidates for whom the model cannot make confident predictions, the model can gather additional data and evolve to become more faithful to the real world.

Similarly, randomness is often necessary when training machine learning models. Models may become too specialized or specific to the data used for training, a problem called "overfitting." Making random choices during the model's training process can prevent this problem. Likewise, models may find a decision rule is well-suited for some portion of the input, but not the best rule overall. Randomness can also help avoid this bias. Consider, for example, a credit-scoring model trained initially on a biased set of data that underrates the creditworthiness of a minority group. Even if the model is the best possible decision rule for a majority of the population similar to the biased input data, the model may unfairly deny access to credit to members of that minority group. In addition to the discrimination, the use of this model would deny creditors business opportunities with the unfairly rejected individuals. Here again, designing the model to occasionally guess randomly, while tracking expected versus actual performance, could improve the model's faithfulness to the population on which it is actually used rather than the biased population on which it was trained. The information from this injection of randomness can be fed back to the model to improve the accuracy and fairness of the system overall.

151 This literature is divided between the machine learning research community in computer science and the study of optimal decisionmaking in statistics. *See supra* note 63.

2. Fair Machine Learning

One commonly understood way to demonstrate that a decision process is independent of sensitive attributes is to preclude the use of those sensitive attributes from consideration. For example, race, gender, and income may be excluded from a decisionmaking process to assert that the process is "race-blind," "gender-blind," or "income-blind."[152] From a technical perspective, however, this approach is naive. Blindness to a sensitive attribute has long been recognized as an insufficient approach to making a process fair. The excluded or "protected" attributes can often be implicit in other nonexcluded attributes. For example, even when race is excluded as a valid criterion for a credit decision, redlining may occur when a zip code is used as proxy that closely aligns with race.[153]

This type of input "blindness" is insufficient to assure fairness and compliance with substantive policy choices. Although there are many conceptions of what defines fairness, we consider here a definition of fairness in which similarly situated people are given similar treatment—that is, a fair process will give similar participants a similar probability of receiving each possible outcome. This is the core principle of a developing literature on *fair classification in machine learning*, an area first formalized by Dwork, Hardt, Pitassi, Reingold, and Zemel.[154] This work stems from a longer line of research on mechanisms for data privacy.[155] We further describe the relationship between fairness in the use of data and privacy below.

The principle that similar people should be treated similarly is often called *individual fairness*, and it is distinct from *group fairness* in the sense that a process can be fair for individuals without being fair for groups.[156] Although it is almost certainly more policy-salient, group fairness is more difficult to define and achieve. The most commonly studied notion of group fairness is *statistical parity*, the idea that an equal fraction of each group should receive

---

152 *See, e.g.*, 12 C.F.R. § 1002.5(b) (2015) ("A creditor shall not inquire about the race, color, religion, national origin, or sex of an applicant or any other person in connection with a credit transaction."); *id.* § 1002.6(b)(9) ("[A] creditor shall not consider race, color, religion, national origin, or sex (or an applicant's or other person's decision not to provide the information) in any aspect of a credit transaction.").

153 *See* Jessica Silver-Greenberg, *New York Accuses Evans Bank of Redlining*, N.Y. TIMES: DEALBOOK (Sept. 2, 2014), http://dealbook.nytimes.com/2014/09/02/new-york-set-to-accuse-evans-bank-of-redlining [https://perma.cc/3YFA-6N4J] (detailing a redlining accusation in great detail).

154 Dwork et al., *supra* note 140.

155 Specifically, the work of Dwork et al. is a generalization of ideas originally presented in Cynthia Dwork, *Differential Privacy*, 2006 PROC. 33RD INT'L COLLOQUIUM ON AUTOMATA, LANGUAGES & PROGRAMMING 1. As discussed below, fairness can be viewed as the property that sensitive or protected status attributes cannot be inferred from decision outcomes, which is very much a privacy property.

156 Sometimes, a more restrictive notion of individual fairness implies group fairness. *Id.* Intuitively, this is because if people who are sufficiently similar are treated sufficiently similarly, there is no way to construct a minority of people who are treated in a systematically different way.

each possible outcome. While statistical parity seems like a desirable policy because it eliminates redundant or proxy encodings of sensitive attributes, it is an imperfect notion of fairness. For example, statistical parity says nothing about whether a process addresses the "right" subset of a group. Imagine an advertisement for an expensive resort: we would not expect that *showing* the advertisement to the same number of people in each income bracket would lead to the same number of people *clicking* on the ad or *buying* the associated product. For example, a malicious advertiser wishing to exclude a minority group from a resort could design its advertising program to *maximize* the likelihood of conversion for the desired group while *minimizing* the likelihood that the ad will result in a sale to the disfavored group. In the same vein, if a company aimed to improve the diversity of its staff by offering the same proportion of interviews to candidates with minority backgrounds as are minority candidate applications, that is no guarantee that the number of people hired will reflect the population of applicants or the population in general. And the company could hide discriminatory practices by inviting only unqualified members of the minority group to apply, effectively creating a self-fulfilling prophecy for decision rules established by machine learning.

The work of Dwork et al. identifies an additional interesting problem with the "fairness through blindness" approach: by remaining blind to sensitive attributes, a classification rule can select exactly the opposite of what is intended.[157] Consider, for example, a system that classifies profiles in a social network as representing either real or fake people based on the uniqueness of their names. In European cultures, from which a majority of the profiles come, names are built by making choices from a relatively small set of possible first and last names, so a name which is unique across this population might be suspected to be fake. However, other cultures (especially Native American cultures) value unique names, so it is common for people in these cultures to have names that are not shared with anyone else. Since a majority of accounts will come from the majority of the population, for which unique names are rare, any classification based on the uniqueness of names will inherently classify real minority profiles as fake at a higher rate than majority profiles,[158] and may also misidentify fake profiles using names drawn from the minority population as real. This unfairness could be remedied if the system were "aware" of the minority status of a name under consideration, since then the

157 *See* Dwork et al., *supra* note 140.

158 That is, the minority group will have a higher false positive rate.

algorithm could know whether the implication of a unique name is that a profile is very likely to be fake or very likely to be real.[159]

This insight explains why the approach taken by Dwork et al. is to enforce similar probabilities of each possible outcome on similar people, requiring that the aggregate difference in probability of any individual receiving any particular outcome be limited.[160] Specifically, Dwork et al. require that this difference in chance of outcome be less than the difference between individuals subject to classification.[161] This requires a mathematically precise notion of how "different" people are, which might be a score of some kind or might naturally arise from the data in question.[162] This notion of similarity must also capture all relevant features, including possibly sensitive or protective attributes such as minority status, gender, or medical history. Because this approach requires the collection and explicit use of sensitive attributes, the work describes its definition of fairness as fairness through awareness.[163] While the work of Dwork et al. provides only a theoretical framework for building fair classifiers, others have used it to build practical systems that perform almost as well as classifiers that are not modified for fairness.

The work of Dwork et al. also provides the theoretical basis for a notion of *fair affirmative action*, the idea that imposing an external constraint on the number of people from particular subgroups who are given particular classifications should have a minimal impact on the principle that similar people are treated similarly. This provides a technique for forcing a fairness requirement such as statistical parity even when it will not arise naturally from some classifier.

A more direct approach to making a machine learning process fair is to modify or select the input data in such a way that the output satisfies some fairness property. For example, in order to make sure that a classifier does not over-reflect the minority status of some group, we could select extra training samples from that group or duplicate samples we already have. In either case,

---

159 In this case, differential treatment based on a protected status attribute *improves* the performance of the automated decision system in a way that *requires* that the system know and make use of the value of that attribute.

160 *See* Dwork et al., *supra* note 140, at 215 (explaining that fairness can be captured under the principle that "two individuals who are similar *with respect to a particular task* should be classified similarly").

161 This is formalized as the proposition that the difference in probability distributions between outcomes for each subgroup of the population being classified is less than the difference between those groups, for a suitable measurement of the difference between groups. For technical reasons, this particular formulation is mathematically convenient, although different bounds might also be useful. For the formal mathematical definition, see *id.* at 216.

162 For example, if the physical location of subjects is a factor in classification, we might naturally use the distance between subjects as one measure of their similarity.

163 Dwork et al., *supra* note 140, at 215.

care must be taken to avoid biasing the training process in some other way or overfitting the model to the nonrepresentative data.

Other work focuses on *fair representations* of data sets. For example, we can take data points and assign them to *clusters*, or groups of close-together points, treating each cluster as a prototypical example of some portion of the original data set. This is the approach taken by Zemel, Wu, Swersky, Pitassi, and Dwork.[164] Specifically, Zemel et al. show how to generate such prototypical representations automatically and in a way that guarantees statistical parity for any subgroup in the original data. In particular, the probability that any person in the protected group is mapped to any particular prototype is equal to the probability that any person not from the protected group is mapped to the same prototype.[165] Therefore, classification procedures which have access only to the prototypes must necessarily not discriminate, since they cannot tell whether the prototype primarily represents protected or unprotected individuals. Zemel et al. test their model on many realistic data sets, including the Heritage Health Prize data set, and determine that it performs nearly as well as best-of-breed competing methods while ensuring substantial levels of fairness.[166] This technique allows for a kind of "fair data disclosure," in which disclosing only the prototypes allows any sort of analysis, fair or unfair, to be run on the data set to generate fair results.

A related approach is to use a technique from machine learning called *regularization*, which involves modifying the model training process to yield models that are more generalizable by introducing a penalty associated with undesirable model attributes or behaviors. This approach has also led to many useful modifications to standard tools in the machine learning repertoire, yielding effective and efficient fair classifiers.[167]

The work of Zemel et al. suggests a related approach, which is also used in practice: the approach of generating *fair synthetic data*. Given any set of data, we can generate new data such that no classifier can tell whether a randomly chosen input was drawn from the real data or the fake data. Furthermore, we can use approaches like that of Zemel et al. to ensure that the new data are at once representative of the original data and also fair for individuals or subgroups. Because synthetic data are randomly generated, they are useful in situations where training a classifier on real data would create privacy concerns. Also, synthetic data can be made public for others to

164 Richard Zemel et al., *Learning Fair Representations*, 28 PROC. 30TH INT'L CONF. ON MACHINE LEARNING 325 (2013).

165 *Id.*

166 *Id.*

167 *See, e.g.*, Toshihiro Kamishima et al., *Fairness-Aware Learning Through Regularization Approach*, 2011 PROC. 3RD IEEE INT'L WORKSHOP ON PRIVACY ASPECTS DATA MINING 643 (describing a model in which two types of regularizers were adopted to enforce fair classification).

use, although care must be taken to avoid allowing others to infer facts about the underlying real data. Such *model inversion* attacks[168] have been demonstrated in practice, along with other *inference* or *deanonymization* attacks that allow sophisticated conclusions without direct access to the actual data that give rise to the conclusions.[169]

All of these approaches demonstrate that it is possible to build a wide range of definitions of fairness into a wide variety of data analysis and classification systems, at least to the extent that a definition of fairness is known or can be approximated in advance. There are no bright-line rules that allow the designer or operator of a machine learning system to guarantee that the system's behavior is compliant with antidiscrimination laws. Nor do we believe that such rules can or even should exist. It is not for technologists to define an ex ante notion of fairness that applies in all cases or even just for a specific system. Rather, fairness must be determined contextually and often must be reviewed ex post. Regardless, it is certainly not impossible to build fairness into automated decision systems, which shows that unconstrained use of data analysis is not always necessary. Uses of data that do not employ methods to investigate or ensure fairness must account for their decision policies in some other way.

Many of these approaches rely on the insufficient notion of group fairness by statistical parity. To the extent that more technical research can help to address the problem of unfairness in big data analysis, it is by expanding the repertoire of definitions of group fairness that can be usefully applied in practice and by providing better exploratory and explanatory tools for comparing different notions of fairness. From a public policy perspective, it would be extremely useful to system designers to have a set of rules, standards, or best practices that explain what notions of fairness should be used in specific real-world applications.

A complementary notion to machine learning systems that can guarantee prespecified, formal fairness properties is the work of Rudin on machine learning systems that are *interpretable*.[170] Such systems generate models that can be used to classify individuals, but also explanations for why those

168 *See* Matthew Fredrikson et al., *Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing*, 2014 PROC. 23RD USENIX SECURITY SYMP. 17 (describing privacy risks in which attackers can predict a patient's genetic markers if provided with the model and some demographic information).

169 For an overview of these techniques, see Arvind Narayanan & Edward W. Felten, No Silver Bullet: De-identification Still Doesn't Work (July 9, 2014), http://randomwalker.info/publications/no-silver-bullet-de-identification.pdf [https://perma.cc/VT2G-7ACG], and Arvind Narayanan et al., A Precautionary Approach to Big Data Privacy (Mar. 19, 2015), http://randomwalker.info/publications/precautionary.pdf [https://perma.cc/FQR3-2MM2].

170 Cynthia Rudin, *Algorithms for Interpretable Machine Learning*, 2014 20TH ACM SIGKDD CONF. ON KNOWLEDGE DISCOVERY & DATA MINING 1519.

classifications were made. These explanations can be reviewed later to understand why the model behaves a certain way, and in some cases how changes in the input data would affect the model's decision. These explanations can be extremely valuable to experts and oversight authorities, who wish to avoid treating models as black boxes.

3. Discrimination, Data Use, and Privacy

A different way to define whether a classification is fair is to say that we cannot tell from the outcome whether the subject was a member of a protected group or not. That is, if an individual's outcome does not allow us to predict that individual's attributes any better than we could by guessing them with no information, we can say that outcome was assigned fairly. To see why this is so, observe the contrary: if the fact that an individual was denied a loan from a particular bank tells you that this individual is more likely to live in a certain neighborhood, this implies that you hold a strong belief that the bank denies credit to residents of this neighborhood and hence a strong belief that the bank makes decisions based on factors other than the objective credit risk presented by applicants.

Thus, fairness can be seen as a form of an *information hiding* requirement similar to privacy. If we accept that a fair decision does not allow us to infer the attributes of a decision subject, we are forced to conclude that fairness is protecting the privacy of those attributes.

Indeed, it is often the case that people are more concerned that their information is used to make some decision or classify them in some way than they are that the information is known or shared. This concern relates to the famous conception of privacy as the "right to be let alone," in that generally people are concerned with the idea that disclosure interrupts their enjoyment of an "inviolate personality."[171]

Data use concerns also surface in the seminal work of Solove, who refers to concerns about "exclusion" in "information processing," or the lack of disclosure to and control by the subject of data processing and "distortion" of a subject's reputation by way of "information dissemination."[172] Solove argues that these problems can be countered by giving subjects knowledge of and control over their own data.[173] In this framework, the predictive models of automated systems, which might use seemingly innocuous or natural behaviors as inputs, create anxiety on the part of data subjects. We propose a complementary approach: if a system's designer can prove to an oversight

171 Samuel D. Warren & Louis D. Brandeis, *The Right to Privacy*, 4 HARV. L. REV. 193, 205 (1890).

172 Daniel J. Solove, *A Taxonomy of Privacy*, 154 U. PA. L. REV. 477, 521, 546 (2006).

173 *See id.* at 546 (detailing privacy statutes that allow individuals to access and correct information that is maintained by government agencies).

entity or to each data subject that the sorts of behaviors that cause these anxieties are simply not possible behaviors of the system, then the use of these data will be more acceptable.

We can draw an analogy between data analysis and classification problems and the more familiar data aggregation and querying problems that are much discussed in the privacy literature. Decisions about an individual represent (potentially private) information about that individual (i.e., one might infer the input data from the decision or the decision itself may be of a private nature), and this raises concerns for privacy. In essence, privacy may be at risk from an automated decision that reveals sensitive information just like fairness may be at risk from an automated decision. In this analogy, a vendor or agency using a model to draw automated decisions wants those decisions to be as accurate as possible, corresponding to the idea in privacy that it is the goal of a data analyst to build as complete and accurate a picture of the data subject as is feasible.

A naive approach to making a data set private is to delete "personally identifying information" from the data set. This is analogous to the current practice of making data analysis fair by removing protected attributes from the input data. However, both approaches fail to provide their promised protections.[174] The failure in fairness is perhaps less surprising than it is in privacy—discrimination law has known for decades about the problem of proxy encodings of protected attributes and their use for making inferences about protected status that may lead to adverse, discriminatory effects.[175]

The work of Hardt[176] relates the work on fairness by Dwork et al.[177] to the work on *differential privacy* by Dwork.[178] As differential privacy is a well-founded notion of protection against inferences and the recovery of an individual identity from "anonymous" data, so are formal fairness properties for automated decision systems sound notions of fairness for individuals and a theoretical framework on which to ground more complicated notions of fairness for protected groups.

---

174 Reidentification of individuals based on inferences from disparate data sets is a growing and important concern that has spawned a large literature in both computer science and law. *See* Ohm, *supra* note 71, at 1704 (arguing that developments in computer science demonstrate that "*[d]ata can be either useful or perfectly anonymous but never both*," and that such developments should "trigger a sea change" in legal scholarship).

175 For example, the law explicitly forbids the (sole) use of certain attributes that are likely to be highly correlated with protected status categories, as in protections against redlining. *See, e.g.*, 12 C.F.R. § 1002.5(b) (2015) ("A creditor shall not inquire about the race, color, religion, national origin, or sex of an applicant or any other person in connection with a credit transaction."); *id.* § 1002.6(b)(9) ("[A] creditor shall not consider race, color, religion, national origin, or sex (or an applicant's or other person's decision not to provide the information) in any aspect of a credit transaction.").

176 Hardt, *supra* note 139.

177 Dwork et al., *supra* note 140.

178 Dwork, *supra* note 155.

The many techniques of building fair data analysis and classification systems described above mostly require decisionmakers to have access to protected status information, at least during the design phase of the algorithm. However, in many cases, concerns about misuse, reuse, or abuse of this information have led to a policy regime where decisionmakers are explicitly barred from using such information. The deployment of these technical tools would require a policy change.[179] The techniques described above could be used to make such a change less prone to engendering the very real concerns of data abuses that have led to the current regime.

C. *Antidiscrimination Law and Algorithmic Decisionmaking*

The goal of Part II—procedural regularity—is relatively simple from a legal standpoint. Procedural regularity is a core idea behind due process: the state cannot single out an individual for a different procedure.[180] An argument that governance measures ensuring algorithmic procedural regularity are *required* by due process is more tenuous,[181] but an agency that implements such measures will not risk violating a legal requirement.

In contrast, governance of algorithms to promote nondiscrimination runs into the complicated field of antidiscrimination law. Here, the movement toward a colorblind interpretation of equal protection has created friction with the precedents involving disparate impact. We argue that, given the current state of antidiscrimination law, designing for nondiscrimination is important because users of algorithms may be legally barred from revising processes to correct for discrimination after the fact, and technical tools offer solutions to help.

1. *Ricci v. DeStefano*: The Tensions Between Equal Protection, Disparate Treatment, and Disparate Impact

Antidiscrimination law is based upon both the constitutional guarantee of equal protection[182] and supplemental statutory protections. Modern interpretations

---

179 One example is the privacy regime created by the Health Insurance Portability and Accountability Act, *see supra* note 78, which forbids the disclosure of certain types of *covered information* beyond those for which the data subject was previously given notice and which limits disclosure to *covered entities* subject to the same restrictions.

180 *See, e.g.*, Arthur S. Miller, *An Affirmative Thrust to Due Process of Law?*, 30 GEO. WASH. L. REV. 399, 403 (1962) ("Procedural due process ('adherence to procedural regularity'), as we have often been told by Supreme Court justices, is the very cornerstone of individual liberties.").

181 *See* Citron, *supra* note 6, at 1278-1300 (arguing that current procedural protections are inadequate for automated decisionmaking).

182 *See* U.S. CONST. amend. XIV, § 1 ("No State shall . . . deny to any person within its jurisdiction the equal protection of the laws."). The Equal Protection Clause has also been interpreted

of the Equal Protection Clause generally have been divided into two camps: those who believe in a color-blind Constitution—protecting individualized assessments and eschewing any evaluations based on group status—and those who support antisubordination attempts to remedy inequalities between groups.[183] The general trend has been toward colorblindness.[184]

For statutory measures, we will focus on Title VII of the Civil Rights Act of 1964.[185] Under Title VII, remedies are available for disparate treatment—discriminatory intent or the formal application of different rules to people of different groups—and disparate impact—results that differ for different groups.[186] Algorithmic decisionmaking blurs the definitions of disparate treatment and disparate impact and poses a number of open questions.[187]

Is it disparate treatment when the inputs used are a proxy for membership in a protected class? Different rules are effectively applied to different groups in this case, but that difference may have no effect on the outcomes.[188] If the people responsible for a decision *know* that an algorithm behaves in a way that has disparate impact, does that mean that they *intend* a discriminatory result?[189] If an algorithm generates poor outcomes for a group of people, how accurate does the algorithm need to be (and how carefully does the decisionmaker need to test alternative algorithms) before the decisionmaker

---

to apply to the federal government through the Due Process Clause of the Fifth Amendment. *See, e.g.*, Kenji Yoshino, *The New Equal Protection*, 124 HARV. L. REV. 747, 748 n.10 (2010).

183 *See, e.g.*, Reva B. Siegel, *From Colorblindness to Antibalkanization: An Emerging Ground of Decision in Race Equality Cases*, 120 YALE L.J. 1278, 1281 (2011) (describing this binary as the common interpretation of equal protection jurisprudence).

184 *See The Supreme Court, 2008 Term—Leading Cases*, 123 HARV. L. REV. 153, 289 (2009) ("The Court's conception of equal protection turns largely on its swing voter, Justice Kennedy, who appears to support a moderate version of the colorblind Constitution."). *But see* Reva B. Siegel, *The Supreme Court, 2012 Term—Foreword: Equality Divided*, 127 HARV. L. REV. 1, 6 (2013) (agreeing that "[s]hifts in equal protection oversight . . . are continuing to grow" but arguing that these changes are "neither colorblind nor evenhanded" because "the Court has encouraged majority claimants to make discriminatory purpose arguments about civil rights law based on inferences the Roberts Court would flatly deny if minority claimants were bringing discriminatory purpose challenges to the criminal law").

185 42 U.S.C. §§ 2000e–e-17 (2012). Title VII applies to employment discrimination on the basis of race, national origin, gender, and religion. The disparate impact framework is also used for discrimination in housing, employment, public entity, public accommodation, and telecommunications (with respect to people with disabilities). *See* Tex. Dep't of Hous. & Cmty. Affairs v. Inclusive Cmtys. Project, Inc., 135 S. Ct. 2507 (2015) (holding that disparate impact claims are cognizable under the Fair Housing Act); Lopez v. Pac. Mar. Ass'n, 657 F.3d 762 (9th Cir. 2011) (deciding a disparate impact claim brought under the Americans with Disabilities Act).

186 *See* Richard Primus, *The Future of Disparate Impact*, 108 MICH. L. REV. 1341, 1350-51 & n.56 (2010) (describing the evolution of the "disparate impact" and "disparate treatment" terminology, and the types of discrimination they are associated with).

187 *See* Barocas & Selbst, *supra* note 8, at 694-714 (noting the ways in which algorithmic data mining techniques can lead to unintentional discrimination against historically prejudiced groups).

188 *Id.* at 695.

189 *Id.* at 700.

can escape disparate impact liability because the factors used are job-related?[190] If, as noted in subsection III.B.2, knowledge of class membership can be used to improve the fairness of outcomes for members of all classes, should doing so be considered disparate treatment?

These doctrines were recently considered in *Ricci v. DeStefano*, in which the Supreme Court held that "before an employer can engage in intentional discrimination for the asserted purpose of avoiding or remedying an unintentional disparate impact, the employer must have a strong basis in evidence to believe it will be subject to disparate-impact liability if it fails to take the race-conscious, discriminatory action."[191] At issue was the City of New Haven's test for firefighter promotions; though the test had been constructed in an attempt to ensure there was no discrimination by race,[192] the pass rates for minorities were about half of the pass rate for whites.[193] The New Haven Civil Service Board did not certify the results of the test (and validate the promotions) due to concerns about fairness and disparate impact liability for the City.[194]

Ricci demonstrates the tension between disparate treatment and disparate impact. Facially neutral policies can produce unequal results for protected classes, but remedying that disparate impact would require the state to treat people differently based on class membership, which *Ricci* forbids. *Ricci* also hints at the difficulties in squaring the Court's move toward a colorblind interpretation of the Equal Protection Clause and the doctrine of disparate impact. The holding does not directly address the constitutional issue, but Justice Scalia's concurrence does note that the "war between disparate impact and equal protection will be waged sooner or later."[195] Both of these doctrinal tensions are of concern to lawmakers and policymakers.

2. *Ricci* Impels Designing for Nondiscrimination

Although *Ricci* has generated wide-ranging conversation about equal protection, disparate treatment, and disparate impact, we wish to emphasize its implications for the governance of decision algorithms for processes where nondiscrimination is a goal. The holding in *Ricci* suggests that we cannot solely rely on auditing for legal reasons in addition to the reasons discussed in Section II.B. If an agency runs an algorithm that has a disparate impact, correcting those results after the fact will trigger the same kind of analysis as

190 *Id.* at 707.
191 557 U.S. 55, 585 (2009).
192 *Id.* at 565.
193 *Id.* at 586-87.
194 *Id.* at 579.
195 *Id.* at 595-96 (Scalia, J., concurring).

New Haven's rejection of its firefighter test results. It is even possible that the Court will "subject some range of disparate impact compliance efforts to strict scrutiny,"[196] a high bar that will be difficult to satisfy in most cases.

The legal difficulties with correcting discriminatory algorithms ex post make measures to design algorithms for nondiscrimination even more important. The Court in *Ricci* took no issue with New Haven's process of designing the test with an eye toward nondiscrimination, reasoning that "Title VII does not prohibit an employer from considering, before administering a test or practice, how to design that test or practice in order to provide a fair opportunity for all individuals, regardless of their race."[197] However, "once that process has been established and employers have made clear their selection criteria, they may not then invalidate the test results, thus upsetting an employee's legitimate expectation not to be judged on the basis of race."[198]

The uneasy fit of algorithmic decisionmaking into the disparate treatment/disparate impact framework does mean that someone could allege disparate treatment because the design of the algorithm includes inputs that are a proxy for class membership, resulting in a formal application of different rules to different groups of people. However, such a claim would be valid against virtually any system with a significant number of inputs. It seems more likely that courts would reject the formal-rule subset of disparate treatment for algorithmic decisions than that they would hold the majority of algorithmic decisionmaking to constitute disparate treatment. In the end, incorporating nondiscrimination in the initial design of algorithms is the safest path that decisionmakers can take, and we should encourage the development and deployment of technical tools to aid in that design.

IV. FOSTERING COLLABORATION ACROSS COMPUTER SCIENCE, LAW, AND POLICY

In this Part, we consider how the types of technological assurance described in previous Parts relate to mechanisms of oversight in law and public policy. In technical approaches, it is traditional to have a detailed, well-defined specification of the behavior of a system for all types of situations. In lawmaking and the application of public policy, it is normal, and even encouraged, for rules to be left open to interpretation, with details filled by human judgment emerging from disputes in specific cases that are resolved after the fact. We offer recommendations for dealing with this apparent mismatch, arguing for greater collaboration between experts in these two different fields.

196 *The Supreme Court, 2008 Term—Leading Cases*, *supra* note 184, at 290.

197 *Ricci*, 557 U.S. at 585.

198 *Id.*

We emphasize that computer scientists cannot assume that the policy process will give them a meaningful, universal, and self-consistent theory of fairness to use as a specification for algorithms. There are structural, political, and jurisprudential reasons why no such theory exists today. Likewise, the policy process would likely not accept such a theory if it were generated by computer scientists.

At the same time, lawmakers and policymakers will need to adapt in light of these new technologies. We highlight changes that stem from automated decisionmaking. First, choices made when designing computer systems embed specific policy decisions and values in those systems whether or not they provide for accountability. Algorithms can, nevertheless, permit direct accountability to the public or to other third parties, despite the fact that full transparency is neither sufficient nor always necessary for accountability. For both groups, we note that the interplay between these areas will raise new questions and may generate new insights into what the goals of these decisionmaking processes should be.

A. *Recommendations for Computer Scientists: Design for After-the-Fact Oversight*

Computer scientists may tend to think of accountability in terms of compliance with a detailed specification set forth before the creation of an algorithm. For example, it is typical for programmers to define bugs based on the specification for a program—anything that differs from the specification is a bug; anything that follows it is a feature.[199]

This Section is intended to inform computer scientists that no one will remove all of the ambiguities and offer them a clear, complete specification. Although lawmakers and policymakers can offer clarifications or other changes to guide the work done by developers,[200] drafters may be unable to remove certain ambiguities for political reasons or be unwilling to resolve details to meet flexibility objectives. As such, computer scientists must account for the lack of precision—and the corresponding need for after-the-fact oversight by courts or other reviewers—when designing decisionmaking algorithms.

A computer scientist's mindset can conflict deeply with many sources of authority to which developers may be responsible. Public opinion and social

---

199 *See, e.g.*, Michael Dubakov, *Visual Specifications*, MEDIUM (Oct. 26, 2013), https://medium.com/@mdubakov/visual-specifications-1d57822a485f [https://perma.cc/SE46-6B2C] ("No specs? No bugs."); SF, *What Is the Difference Between Bug and New Feature in Terms of Segregation of Responsibilities?*, STACKEXCHANGE (July 12, 2011, 6:51 AM), http://programmers.stackexchange.com/questions/92081/what-is-the-difference-between-bug-and-new-feature-in-terms-of-segregation-of-re [https://perma.cc/PPM6-HFAA] ("You could put an artificial barrier: if it's against specs, it's a bug. If it requires changing specs . . . it's a feature.").

200 *See infra* Section IV.B.

norms are inherently not precisely specified. The corporate requirements to satisfy one's supervisor (or one's supervisor's supervisor) may not be clear. Perhaps most importantly and least intuitively for computer scientists, the operations of U.S. law and public policy also work against clear specifications. These processes often deliberately create ambiguous laws and guidance, leaving details—or sometimes even major concepts—open to interpretation.[201]

One cause of this ambiguity is the political reality of legislation. Legislators may be unable to gather majority support to agree on the details of a proposed law, but may be able to get a majority of votes to pass relatively vague language that leaves various terms and conditions unspecified.[202] For example, different legislators may support conflicting specific proposals that can be encompassed by a more general bill.[203] Even legislators who do not know precisely what they want may still object to a particular proposed detail; each detail that caused sufficient objections would need to be stripped out of a bill before it could become law.

Another explanation of ambiguity is that legislators may have uncertainty about the situations to which a law or policy will apply. Drafters may worry that they have not fully considered all of the possibilities. This creates an incentive to build in enough flexibility to cover unexpected circumstances that currently exist or may exist in the future.[204] The U.S. Constitution is often held up as a model in this regard: generalized provisions for governance and individual rights continue to be applicable even as the landscape of society changes dramatically.[205]

Finally, ambiguity may stem from shared uncertainty about how best to solve even a known problem. Here, drafters may feel that they know what

---

201 *See, e.g.*, Marbury v. Madison, 5 U.S. (1 Cranch) 137 (1803) (establishing the practice of judicial review, on which the Constitution was silent); *see also* 47 U.S.C. § 222(c)(1) (2012) (requiring a telecommunications carrier to get the "approval of the customer" to use or disclose customer proprietary network information, and requiring the Federal Communications Commission to define "approval").

202 *See* Victoria F. Nourse & Jane S. Schacter, *The Politics of Legislative Drafting: A Congressional Case Study*, 77 N.Y.U. L. REV. 575, 593 (2002) ("Several staffers thought that pressures of time, and the political imperative to get a bill 'done,' bred ambiguity. Indeed, one staffer emphasized that while it was well and good to draft a bill clearly, there was no guarantee that the clear language would be passed by the House or make it through conference.").

203 *See* Richard L. Hasen, *Vote Buying*, 88 CALIF. L. REV. 1323, 1339 (2000) (describing the practice of "legislative logrolling").

204 *See, e.g.*, 17 U.S.C. § 1201 (2012) (granting the Copyright Office the power to create exemptions from the statute's prohibition on anti-circumvention).

205 *See* DAVID A. STRAUSS, THE LIVING CONSTITUTION (2010). Laws governing law enforcement access to personal electronic records are often cited as a counterexample, with over-specific provisions in the Electronic Communications Privacy Act, 18 U.S.C. §§ 2510–2704 (2012), that fail to account for a shift in technology to a regime where most records reside with third party service providers, not users' own computers. For a more detailed explanation, see Orin S. Kerr, *Applying the Fourth Amendment to the Internet: A General Approach*, 62 STAN. L. REV. 1005 (2010).

situations will arise but still not know how they want to deal with them. They may, in effect, choose to delegate authority to other parties by underspecifying particular aspects of a law or policy. Vagueness supports experimentation to help determine what methods are most effective or desirable.[206]

The United States has a long history of dealing with these ambiguities through after-the-fact and retroactive oversight by the courts.[207] In our common law system, ambiguities and uncertainties are left unaddressed until there is a dispute and their resolution becomes necessary. Disagreements about the application of a law or regulation to a specific set of facts are resolved through cases, and the areas of ambiguity are clarified over time by the accretion of many rulings on specific situations.[208] Even when statutes and regulations may have specific and detailed language, they are interpreted through cases—with extensive deference often given to the expertise of administrative agencies.[209] Those cases form binding precedents that, in the U.S. common law system, are an additional source of legal authority alongside the statutes themselves.[210] The gradual development and extension of law and regulations through cases with specific fact patterns allows for careful consideration of meaning and effects at a level of granularity that is usually impossible to reach during the drafting process.[211]

In practice, these characteristics imply that computer scientists should focus on creating algorithms that are reviewable, not just compliant with the specifications that are generated in the drafting process.[212] For example, this means it would have been good for the Diversity Visa Lottery described in Section II.E to use an algorithm that made fair, random choices and it would be desirable for the State Department to be able to demonstrate that property to a court or a skeptical lottery participant.[213]

---

206 A similar logic—policy experimentation among the states—is one of the principles underlying federalism. *See* New State Ice Co. v. Liebmann, 285 U.S. 262, 311 (1932) (Brandeis, J., dissenting) (praising the ability of a state to "serve as a laboratory" for democracy).

207 *See generally* E. ALLAN FARNSWORTH, AN INTRODUCTION TO THE LEGAL SYSTEM OF THE UNITED STATES (Steve Sheppard ed., 4th ed. 2010).

208 *See generally id.*

209 *See* Chevron U.S.A. Inc. v. Nat. Res. Def. Council, Inc., 476 U.S. 837 (1984).

210 *See generally* FARNSWORTH, *supra* note 207.

211 *Id.*

212 Another possible conclusion is that certain algorithms should also be developed to be flexible, permitting adaptation as new cases, laws, or regulations add to the initial specifications. The need to adapt algorithms is discussed further in subsection IV.B.1. This also reflects the current insufficiency of building a system in accord with a particular specification, though oversight or enforcement bodies evaluating the decision at a later point in time will still need to be able to certify compliance with any actual specifications.

213 Algorithms offer a new opportunity for decisionmaking processes to be reviewed by nontraditional overseers: decision recipients, members of the public, or even concerned nongovernmental organizations. We discuss this possibility further in subsection IV.B.2.

The technical approaches described in this Article[214] provide several ways for algorithm designers to ensure that the actual basis for a decision can be verified later. With these tools, reviewers can check whether an algorithm actually was used to make a particular decision,[215] whether random inputs were chosen fairly,[216] and whether the algorithm comports with certain principles specified at the time of the design.[217] Essentially, these technical tools allow continued after-the-fact evaluations of algorithms by allowing for and assisting the judicial system's traditional role in ultimately determining the legality of particular decisionmaking.[218]

Implementing the approaches described in this Article would improve the accountability of decisionmaking algorithms dramatically, but we see that implementation as only a first step. We encourage research into extensions of these technical tools, as well as new techniques designed to facilitate oversight.

B. *Recommendations for Lawmakers and Policymakers*

The other side of the coin is that lawmakers and policymakers need to recognize and adapt to the changes wrought by algorithmic decisionmaking. Characteristics of algorithms offer both new opportunities and new challenges for the development of legal regimes governing decisionmaking: algorithmic decisionmaking can reduce the benefits of ambiguity, increase accountability to the public, and permit greater accountability than was previously possible in cases where aspects of the decision process remain secret.

1. Reduced Benefits of Ambiguity

Although computer scientists can build algorithms to permit after-the-fact assessment and accountability, they cannot alter the fact that any algorithm design will encode specific values and involve specific rules. Furthermore, the design of a computer system may limit opportunities for after-the-fact accountability. In other words, if a system is not designed to permit certification of a particular characteristic, an oversight body cannot be certain that it will be able to certify that characteristic. Both of these traits imply that automated decisionmaking can exacerbate certain disadvantages of legal ambiguities.

In the framework set forth above,[219] we identify key drivers of ambiguity: political stalemate, uncertainty about future circumstances, and desire for

214 *See supra* Sections II.B, III.B.

215 *See supra* Section II.C.

216 *See supra* subsection II.C.4.

217 *See supra* subsection II.C.1.

218 Computer scientists model this after-the-fact input as an "oracle" that can be consulted only rarely on the acceptability of the algorithm. *See* Kroll, *supra* note 118.

219 *See supra* Section IV.A.

policy experimentation. Here, with respect to each of these drivers, we will discuss how the shift to algorithmic decisionmaking diminishes the appeal of ambiguity, and we will suggest ways of retaining the functional benefits that ambiguity provides in the U.S. lawmaking system and ways that are more amenable to automation.

Ambiguity stemming from political stalemate essentially passes the buck for determining details from legislators to someone later on in the process. These later actors tend to be more sheltered from political pressures and thus able to make specific decisions without risking their jobs at the next election. Judges and administrative agencies frequently fill this role. Courts are expected to offer impartial decisions resistant to public pressure,[220] and administrative agencies are expected to retain staff who offer subject-matter expertise beyond what is expected of legislators, despite changes in political administrations.[221]

However, this transfer of responsibility often works in less than ideal ways when it comes to software systems.[222] Fully automated decisionmaking may exacerbate these problems by adding another actor to whom the responsibility can devolve: the developer who programs the decisionmaking software. Citron offers examples of failures in automated systems that determine benefits eligibility, notably the airport "No Fly" lists, terrorist identifications, and punishment for "dead-beat" parents.[223] Lawmakers should consider this possibility and avoid giving the responsibility for filling in the details of the law to program developers because (1) the algorithms will apply broadly, affecting all participants; (2) the program developer is unlikely to be held accountable by the current political process; and (3) the program developer is unlikely to have substantive expertise about the political decision being made.[224]

---

220 *See, e.g.*, THE FEDERALIST NO. 78 (Alexander Hamilton) (laying out the philosophy that the judiciary's role is to secure an "impartial administration of the laws"). However, the rise of elected judges raises questions about this traditional role of the court system. *See* Stephen J. Choi et al., *Professionals or Politicians: The Uncertain Empirical Case for an Elected Rather Than Appointed Judiciary* (Univ. of Chi. Law Sch., John M. Olin Law & Economics Working Paper No. 357, 2007) (finding that elected judges behave more like politicians than appointed independent judges).

221 This is the rationale of the *Chevron* doctrine of judicial deference to administrative agency actions. Chevron U.S.A. Inc. v. Nat. Res. Def. Council, Inc., 476 U.S. 837 (1984).

222 For example, Citron argues that "[d]istortions in policy have been attributed to the fact that programmers lack 'policy knowledge,'" and that this leads to software that does not reflect policy goals. Citron, *supra* note 6, at 1261. Ohm also reports on a comment of Felten that "[i]n technology policy debates, lawyers put too much faith in technical solutions, while technologists put too much faith in legal solutions." Paul Ohm, *Breaking Felten's Third Law: How Not to Fix the Internet*, 87 DENV. L. REV. ONLINE (2010), http://www.denverlawreview.org/how-to-regulate/2010/2/22/breaking-feltens-third-law-how-not-to-fix-the-internet.html [https://perma.cc/6RGQ-KUMW] (internal quotation marks omitted).

223 Citron, *supra* note 6, at 1256-57.

224 *Id.* at 1254-55. A distinction should be drawn here between the responsibilities given to individual developers of particular algorithms and the responsibilities given to computer scientists

One potential method for restricting the discretion of developers without requiring specifications in the legislation itself would be for administrative agencies to publish guidance for software development. Difficulties in translating between code choices and policy effects still would exist, but they could be partly eased using the technical methods we have described.[225] For example, administrative agencies could work together with developers to identify the properties they want a piece of software to possess, and the program could then be designed to satisfy those properties and permit proof.

Ambiguity generated by uncertainty about the situational circumstances or ambiguity motivated by a desire for policy experimentation presents a more difficult concern. Here, the problem raised by automated decisionmaking is that a piece of software locks in a particular interpretation of law for the duration of its use, and, especially in government contexts, provisions to update the software code may not be made. Worries about changing or unexpected circumstances could be assuaged by adding sunset provisions to software systems,[226] requiring periodic review and reconsideration of the software. Additionally, software should be designed with eventual revisions and updates in mind. As for preserving the benefits of policy experimentation, the traditional solution might be having multiple programs that take multiple approaches deployed simultaneously. A more sophisticated version of this solution is the incorporation of machine learning into decisionmaking systems. Again, machine learning can have its own fairness pitfalls,[227] and care should be taken to consider fair machine learning methods[228] and to build in precautions like persistent testing of the hypotheses built into the machine learning model.[229]

More generally, the benefits of ambiguity decrease in the case of algorithmic decisionmaking. Here, an uninformed programming actor may determine the details and then apply them broadly. In addition, the choice of algorithm cements the particular policy choices encoded in that software for

---

in general. Great gains can be made by improved dialogue between computer scientists and lawmakers and policymakers about how to design algorithms to reach social goals.

225 *See supra* Sections II.B, III.B.

226 The effectiveness of sunset provisions in leading to actual reconsideration and change is debatable. The inertia of the pre-existing choices can be hard to overcome. *See, e.g.*, Mark A. Lemley & David McGowan, *Legal Implications of Network Economic Effects*, 86 CALIF. L. REV. 479, 481-82 (1998) (noting that stare decisis, confusion regarding the role of theory, differing normative values, and other factors impede the progress of the law).

227 *See supra* subsection III.B.1 (noting that machine learning programs give predictions but not confidence levels).

228 *See supra* Sections III.A–B.

229 In other words, even after a machine learning algorithm determines that a particular rule should be used to produce particular results, it always should continue to test inputs that do not follow that rule. *See, e.g.*, RUSSELL & NORVIG, *supra* note 67.

as long as it is used. Drafters should instead consider whether they should increase the specificity offered by law and policy governing these algorithms to prevent coders from filling the ambiguity.

To a certain extent, this question mirrors the rules versus standards debate about the relative merits of laws that specify actions and their repercussions (for example, a speed limit) and those that espouse a principle open to interpretation (for example, "drive at a speed reasonable for the conditions").[230] Rules give clarity and forewarning, while standards offer greater flexibility for interpretation.[231]

Here, the question is whether drafters should include additional and clearer specifications for developers. In practice, drafters may wish to incorporate a set of narrow rules within a broad, overarching standard. For example, drafters could include specifications of each of the properties that they want a piece of software to possess and requirements that the developer design that program in a way that renders those properties provable upon review. Additionally, drafters might consider requiring a general statement of purpose for the algorithm. Doing so would give the developer some flexibility in writing the code while also ensuring that particular properties can be checked later.

2. Accountability to the Public

Oversight is traditionally performed by courts, enforcement agencies, or other designated entities such as government prosecutors.[232] Typically, the public and third parties have an indirect oversight role through the ability to provide political feedback and the ability to bring lawsuits if their specific circumstances allow.[233] The use of software can alter how effectively the legal system and the public can oversee the decisionmaking process.

In one sense, decisionmaking computer systems can enhance accountability to the public and interested third parties by permitting greater involvement in oversight. The technical tools we describe allow for a more direct form of oversight by these parties. Unlike traditional legal oversight mechanisms that generally require discovery or the gathering of internal evidence, the technical tools may enable verifications by the public and by third parties that are

---

230 *See, e.g.*, Louis Kaplow, *Rules Versus Standards: An Economic Analysis*, 42 DUKE L.J. 557, 562-66 (1992) (arguing that rules are more costly to promulgate while standards are more costly on individuals).

231 *See* Kathleen M. Sullivan, *The Supreme Court, 1991 Term—Forward: The Justices of Rules and Standards*, 106 HARV. L. REV. 22, 26 (1992) (explaining the rule versus standard choice in terms of force of precedent, constitutional reading, and formulation of operative tests).

232 *See* FARNSWORTH, *supra* note 207.

233 The public can vote political leaders out of office and aggrieved parties can bring lawsuits to seek vindication.

completely independent from the organizations using the software. For example, technologically proficient members of the public or third parties could verify that a particular algorithm was used in a program or that the program has particular properties. In addition, a system could be built to enable participants to check these properties for their own outcomes so that nontechnical users could verify these facts while the system as a whole would be overseen by others—potentially both inside and outside of government—who have the necessary technological expertise. As another example, third parties could be involved in generating fair randomness.[234]

In contrast to the possibility for enhanced public accountability, the use of software without the reliance on technical tools for oversight, as we have described, can reduce accountability to the public because courts and other policy actors are generally ill-equipped to evaluate software, thereby hampering our traditional scrutiny of decisionmaking. The U.S. court system is designed to protect against wrongful government actions through the power of judicial review.[235] Judicial review gives judges the power and responsibility to determine if government actions comply with legal obligations. Similarly, for private actions, the legal system vests judges and regulatory agencies with the authority to determine whether those actions are consistent with legal standards.

The use of software systems to make decisions, however, shifts these burdens to external experts or to the organizations creating and deploying the software. Courts and enforcement agencies are no longer able to make a determination as to whether the rules have been properly applied or whether fairness obligations have been met. That determination shifts to the experts evaluating the automated decisionmaking process. One way to address this unintended shift in responsibility is to appoint technical experts as special masters. Courts typically appoint special masters to perform functions on behalf of the court that require special skill or knowledge.[236]

Another issue that challenges public accountability is the validation of the technical tools we have described. For courts, technical tools cannot be accepted until their integrity and reliability are proven. Courts have long confronted the problem of the admissibility of scientific evidence. There is a rich literature about the standards courts should use to admit expert scientific

234 *See supra* note 116 (using a quantum source to generate randomness).

235 *See* Marbury v. Madison, 5 U.S. (1 Cranch) 137, 177 (1803) ("It is emphatically the province and duty of the judicial department to say what the law is.").

236 *See, e.g.*, United States v. Microsoft Corp., 147 F.3d 935, 959 n.4 (D.C. Cir. 1998) (noting Larry Lessig's role as a court-appointed special master for technical issues in the antitrust case brought against Microsoft).

evidence, and, even now, federal and state standards vary.[237] The courts, for example, took years during the 1980s and 90s to establish and accept the scientific validity of DNA and the methods used to isolate and test DNA.[238] Even now, there are concerns that some scientific methods (e.g. matching DNA based on mixtures of DNA) may be receiving undeserved deference from courts and thus resulting in faulty findings of fact.[239] Following much debate, the Federal Rules of Evidence spelled out a federal standard for the acceptability of new scientific methods in adversarial proceedings.[240] In 1993, the Supreme Court adjusted those standards to take account of factors that include testing, peer review and publication.[241] These evidentiary standards address the validation of technical tools used to examine automated decisionmaking, but still leave open the assurance of the technical tools' reliability. Ordinarily, the U.S. legal system relies on the adversarial process to assure the accuracy of findings. This attribute may be preserved by allowing multiple experts to test software-driven processes.

3. Secrets and Accountability

Implementing automated decisionmaking in a socially and politically acceptable way requires progress in our ability to communicate and understand fine-grained partial information about how decisions are reached. Full transparency (disclosing everything) is technically trivial but politically and practically infeasible and may not be useful, as described in Section II.A. However, disclosing nothing about the basis for a decision is socially unacceptable and generally poses a technical challenge. Lawmakers and

---

237 *See, e.g.*, Paul C. Giannelli, *The Admissibility of Novel Scientific Evidence:* Frye v. United States, *a Half-Century Later*, 80 COLUM. L. REV. 1197 (1980) (highlighting the development of the standards used for evidentiary scientific evidence); Heather G. Hamilton, Note, *The Movement from* Frye *to* Daubert*: Where Do the States Stand?*, 38 JURIMETRICS 201 (1998) (emphasizing the lack of uniformity of state approaches).

238 *See, e.g.*, NAT'L RESEARCH COUNCIL, THE EVALUATION OF FORENSIC DNA EVIDENCE 166-211 (1996) (discussing the legal implications of the use of forensic DNA testing as well as the procedural and evidentiary rules that affect such use).

239 Logan Koepke, *Should Secret Code Help Convict?*, MEDIUM (Mar. 24, 2016), https://medium.com/equal-future/should-secret-code-help-convict-7c864baffe15#.j9k1cwhoo [https://perma.cc/6LNW-WN6W].

240 *See* FED. R. EVID. 702 ("A witness who is qualified as an expert by knowledge, skill, experience, training, or education may testify in the form of an opinion or otherwise if . . . the expert's scientific, technical, or other specialized knowledge will help the trier of fact to understand the evidence or to determine a fact in issue . . . .").

241 *See* Daubert v. Merrell Dow Pharm., Inc., 509 U.S. 579, 592-95 (1993) (explaining that a judge faced with a proffer of expert scientific testimony must assess whether the testimony's underlying reasoning is valid, and in doing so, consider whether the technique or theory in question can be tested and whether it has been subjected to peer review and publication).

policymakers should remember that it is possible to make an algorithm accountable without the evaluator having full access to the algorithm.[242]

U.S. law and policy often focus on transparency and sometimes even equate oversight with transparency for the overseer.[243] As such, accountability without full transparency may seem counterintuitive. However, oversight based on partial information occurs regularly within the legal system. Courts prevent consideration of many types of information for various policy reasons: disclosures of classified information may be prevented or limited to preserve national security;[244] juvenile records may be sealed because of the notion that mistakes made in one's youth should not follow them forever;[245] and other evidence is deemed inadmissible for a multitude of reasons, including being unscientific,[246] hearsay,[247] inflammatory,[248] or illegally obtained.[249] Thus, all of the rules of evidence could be construed as precedent for the idea that optimal oversight does not require full information.

There are strong policy justifications for holding back information in the case of automated decisionmaking. Revealing software source code and input data can expose trade secrets, violate privacy, hamper law enforcement, or lead to gaming of the decisionmaking process.[250] The advantage of computer systems is that concealment of code and data does not imply an inability to analyze the code and data. The technical tools we describe give lawmakers and policymakers the ability to keep software programs and their inputs secret while still rendering them accountable. They can implement these tools in government-run algorithms, such as the DVL, and incentivize nongovernmental actors to use them, perhaps by mandating use or by requiring transparency—at least to courts—of code and inputs if they do not employ such technical tools.

---

242 *See supra* subsections II.C.3–II.D.

243 *See, e.g.*, 5 U.S.C. § 552 (2012) (requiring agencies to make government records available to the public); 15 U.S.C. § 6803 (2012) (requiring financial institutions to provide annual privacy notices to customers as a transparency measure).

244 *See* 18 U.S.C. § 798(a) (2012) (providing that the disclosure of classified government information may result in criminal liability).

245 *See, e.g.*, N.Y. CRIM. PROC. § 720.15 (requiring filing under seal in juvenile proceedings).

246 *See* FED. R. EVID. 702 (establishing the court's discretion to admit scientific evidence).

247 *See* FED. R. EVID. 802 (stating that hearsay evidence is inadmissible unless a federal statute, the rules of evidence, or the Supreme Court provides otherwise).

248 *See* FED. R. EVID. 403 (providing for the exclusion of relevant evidence for prejudice).

249 *See, e.g.*, 18 U.S.C. § 2515 (2012) (setting an exclusionary rule for evidence obtained through wire tap or interception).

250 *See* Sections II.A–B.

* * * * *