Accountable Algorithms

Joshua Alexander Kroll

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Professor Edward W. Felten

September 2015

Abstract

Important decisions about people are increasingly made by algorithms: Votes are counted; voter rolls are purged; financial aid decisions are made; taxpayers are chosen for audits; air travelers are selected for search; credit eligibility decisions are made. Citizens, and society as a whole, have an interest in making these processes more transparent. Yet the full basis for these decisions is rarely available to affected people: the algorithm or some inputs may be secret; or the implementation may be secret; or the process may not be precisely described. A person who suspects the process went wrong has little recourse. And an oversight authority who wants to ensure that decisions are made according to an acceptable policy has little assurance that proffered decision rules match decisions for actual users.

Traditionally, Computer Science addresses these problems by demanding a specification of the desired behavior, which can be enforced or verified. But this model is poorly suited to real-world oversight tasks, where specifications are complicated or might not be known in advance; laws are often ambiguous precisely because it would be politically (and practically) infeasible to give a precise description of their meaning. People do their best to approximate what they believe the law will allow and disputes about what is acceptable happen after-the-fact via expensive investigation and adjudication (e.g., in a court or legislature). Actual oversight, in which real decisions are reviewed for their correctness, fairness, or faithfulness to a rule happens only rarely, if at all.

Further, Computer Science often sees rules as self-enforcing: the mere fact that an automated check fails is sufficient to demonstrate that some choice was invalid. However, like all rules, automated rules are just the intentions of a system designer and only bear relevance if people will actually follow them, either due to internalized incentives or the external threat of punishment.

This dissertation relates the tools of technology to the problem of overseeing decision making processes. These methods use the tools of computer science to cryptographically ensure the technical properties that can be proven, while providing information necessary for a political, legal, or social oversight process to operate effectively. First, we present an example of the current state-of-the-art in technical systems for accountability: a well-defined policy, specified in advance, is operationalized with technical tools, and those same tools are used to convince outsiders or auditors. Our system enables the accountable execution of legal orders by a judge allowing an investigator compelled access to private records, so that the investigator's access to sensitive information is limited to only that information which the judge has explicitly allowed (and this can be confirmed by a disinterested third party). Moving beyond these methods, we present a general framework for *accountable algorithms*, unifying a suite of cryptographic tools to design processes that enable meaningful after-the-fact oversight, consistent with the norm in law and policy. Accountable algorithms can attest to the valid operation of a decision policy even when all or part of that policy is kept secret.

Acknowledgements

I am extremely grateful to my advisor, Ed Felten, for his unflagging support, his generosity, and his willingness to go out of his way to be available and helpful to me and all of his students. I've learned much from Ed about how to spot interesting research problems, how to execute substantial research, and how to communicate the results of that research for maximum impact.

I have been fortunate to work with a wide range of collaborators on a number of projects, and would like to acknowledge the contributions to my graduate career of (in alphabetical order) Andrew Appel, Dan Boneh, Mitchell Berger, Joseph Bonneau, Nicholas Butowski, Joe Calandrino, Jeremy Clark, Will Clarkson, Ian Davey, Anne Edmundson, Steve Englehardt, Ariel Feldman, Steven Goldfeder, J. Alex Halderman, Sean Hervey-Jumper, Timothy B. Lee, Peter Johnsen, Ben Jones, Seth Josefer, Harry Kalodner, Anna Kornfeld-Simpson, Elliott Krauss, Andrew Miller, Arvind Narayanan, Valeria Nikolaenko, Laura Roberts, Cole Schlesinger, Gordon Stewart, David Wu, Harlan Yu, William Zeller, and Joe Zimmerman. While I did not always get to work with each of you as much as I would have wanted, you certainly all had a strong impact on my thinking about research. I am especially grateful to my fellow students, from whom I learned the most during my time at Princeton.

Academically, I have been strongly influenced by Andrew Appel, who taught me the value of thinking clearly about a problem before diving into a solution while remaining efficient, and by Arvind Narayanan, who taught me the value of looking where others are not and the value of eschewing short-term wins when one believes in a long-term plan. Arvind has also taught me the value of real-world adoption as a measure of research output. I would like to thank my entire thesis committee: Andrew Appel, Nick Feamster, Edward W. Felten, Matthew D. Green, and Arvind Narayanan for their candid, careful, and supportive feedback throughout this process.

I have been very fortunate to have the unflagging support of my wife, Ann Frey Kroll, who gave me the strength to keep believing in my work even when it was difficult or seemed not to be going anywhere. Ann has shown immense patience and has taught me to have courage in the face of even great difficulty.

I am also thankful for the support of my family—my parents, Ron and Darcie Kroll; my sister, Amy Kroll; and the many extended family and in-laws who have inspired me—all of them have enriched my experience in pursuing my research career in their own way.

This dissertation is dedicated to the memory of my grandparents, Arvel Henning Mattson, who helped stir my childhood interest in engineering, and Verna Trom Mattson, who taught me much about the world and how to live in it effectively. Without them, I would not have had the opportunity to pursue the education I have received, and I am truly grateful.

I also acknowledge the memory of my friend, Vice Admiral Thomas J. Hughes, U.S.N., who is responsible for much more of my career than he ever realized.

Additionally, I must take the opportunity to thank Princeton for giving me the opportunity and resources to give so much of my time to research in the years since 2009. I am especially grateful to Joy Montero from the office of the Dean of the Graduate School and her successor Lisa Schreyer for their support through my years in New Jersey.

Finally, the work in this dissertation was supported in part by a National Science Foundation Graduate Research Fellowship under Grant No. DGE-1148900 and by a gift by Norm B. Tomlinson Jr., '48 to the Center for Information Technology Policy at Princeton University.

To my wife, Ann, whose courage and strength impel me to keep trying.

Contents

Chapter 1

Introduction

Computer systems increasingly determine important and consequential decisions once governed only by people: computers count votes; purge voter rolls; choose taxpayers for audits; designate individuals or neighborhoods for law enforcement or intelligence scrutiny; select air travelers for search; grant visas; and decide credit eligibility. The efficiency and accuracy of automated decision making and data analysis ensure that its domain will continue to expand. Yet the full basis for these decisions is rarely available to affected people: the underlying algorithm or some inputs may be secret; or the implementation may be secret; or the process may not be precisely described. A person who suspects the process went wrong has little recourse.

Additionally, 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 for the oversight of human decision makers. Many observers have argued that our current frameworks are not well adapted for situations in which a potentially incorrect, unjustified, or unfair outcome emerges from a computer [294]. Citizens, and society as a whole, have an interest in making these processes more accountable.

As an example of a process that would benefit from increased accountability, consider a government tax authority that is deciding which taxpayers to audit. Taxpayers are worried that audit decisions may be based on bias or political agenda rather than legitimate criteria; or they may be worried that decisions will be incorrect because the authority's software is buggy. The authority does not want to disclose the details of its decision policy, for fear that tax evaders will be able to avoid audits. An oversight body, such as a court or legislature, must be able to evaluate the tax authority's policy after the fact, but ideally can do so without unnecessarily learning the private financial information of taxpayers.

Questions of accountability and fairness in automated decision processes are getting increasing attention in the law and policy literature [26, 97]. A 2014 White House review on big data and public policy stated that "Some of the most profound challenges revealed during this review concern how big data analytics may lead to disparate inequitable treatment, particularly of disadvantaged groups, or create such an opaque decision-making environment that individual autonomy is lost in an impenetrable set of algorithms" [46, p. 10].

We are optimistic, however: although we agree that the growing popularity of automated decision-making in government and industry poses a challenge for existing legal and regulatory frameworks, we believe that adoption of new technologies also provides powerful new opportunities to address many of these concerns. While the current governance of automated decision making is underdeveloped, automated processes can be designed for governance and oversight. Specifically, this dissertation introduces a suite of technical tools which provide the bridge from formal assurances achievable technically to human-level political trust. We call computer systems that employ these methods *accountable algorithms*.[1] A computer system is accountable

[1]The term "algorithm" is assigned disparate technical meaning in the literatures of computer science and other fields. Donald Knuth famously defined algorithms as separate from mathematical formulae in that they must: (i.) "always terminate after a finite number of steps", (ii.) that "each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously

when its behavior comports with the political, legal, and social norms under which it operates and an observer such as a concerned citizen or oversight authority can determine that this is the case.

Consider a company's decision to offer a particular deal to a consumer. This might be a pricing decision, a credit approval, an offer of insurance, or membership in some group.[2] There is a social interest in having such decisions made fairly, especially with respect to the treatment of historically disadvantaged groups (an interest which, in many cases, is enshrined in law [26, 94, 315]). Companies may wish to provide evidence to the public that their methods meet some standard for fairness or procedural regularity, perhaps to rebut inaccurate accusations of unfairness, to head off potential criticism, or to establish public goodwill. But how can even expert observers come to trust this evidence and what evidence must be provided? Because the answers today are often unclear, such evidence is often *ad hoc* and based on promises made by the operator of some computer system under the assumption that violations of these promises, whether deliberate or accidental, will be detected and will result in

specified for each case", (iii.) that input is "quantities which are given to it initially before the algorithm begins", (iv.) that output is "quantities which have a specified relation to the inputs", and (v.) that "all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using paper and pencil" [229]. Similarly and more simply, in their widely used textbook, Cormen, Leiserson, Rivest and Stein define an algorithm is "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" [102].

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 taken 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 [326]. Sandvig argues that even algorithms as simple as sorting "have their own public relations" and are inherently human in their decisions. Another communications scholar, Diakopoulos [121], defines algorithms in the narrow sense (as "as a series of steps undertaken in order to solve a particular problem or accomplish a defined outcome") but considers them in the broad sense (saying "algorithms can arguably make mistakes and operate with biases", which does not make sense for the narrower technical definition). This confusion is common to much of the literature on algorithms and accountability outside of the computer science literature, which we summarize in Chapter 2.

This dissertation adopts the precise definition of the word "algorithm" from computer science and, following Friedman and Nissenbaum [153], refers to the broader concept of an automated system deployed in a social or human context as a "computer system".

[2]Decisions about membership in a particular group take on particular salience in light of the use of machine learning techniques for advanced customer segmentation by many businesses [97].

punishment. Only rarely do systems provide affirmative evidence as to how they are operating and why.

Our framework will allow a decision maker to maintain the secrecy of its policy while allowing each decision subject and an oversight body to verify that the decision maker is accountable for its policy in the following sense.

Definition 1. *Accountability (informal)*

the authority committed to its policy in advance;

the result asserted by the authority is the correct output of the authority's committed policy when applied to the individual taxpayer's data;

any randomness used in the decision was generated fairly; and

the authority can reveal its policy to an oversight body for examination later, without revealing taxpayers' private financial details, and that body as well as taxpayers can verify that the revealed policy is the one that was used to make decisions about them.

We revisit how to define accountability in a way suitable for our purposes in Chapter 2 and provide a more formal model in Chapter 5.

In general, accountability fosters important social values, such as fairness, transparency, and due process, each of which is a deep subject addressed by an enormous literature. Accountable algorithms as we define them can play a role in furthering these goals if used wisely. While our techniques in isolation cannot solve any major social problem, we believe they exemplify necessary tools for doing so. The use of such methods can improve upon not only the current governance of computer systems, but also—in certain cases—the governance of decision-making in general. The implicit (or explicit) biases of human decision-makers can be difficult to find and root out, but decisions based on a formalized rule can be reviewed for fairness, coherence with social values, or compliance with existing or proposed law.

The example of a tax collection authority determining whom to audit provides an excellent illustration of this. A well governed process for determining who gets audited must be trustworthy to those chosen for auditing and also to those not chosen for auditing, so that everyone believes the revenue collection process is fair and unbiased or so that they are willing to comply with the rigors of the audit itself.[3] Whether the process is implemented by a computer or a collection of functionaries, accountability requires the basis for each decision to be available for later review by an oversight body. If decisions are made in the mind of a human, the actual process by which a decision is reached is unavailable to the overseer, who must therefore consider the possibility that the most adverse possible decision process was used (e.g., if the human decision maker was given protected status information about a taxpayer, e.g. that taxpayers race or gender, an overseer could not rule out the possibility that one of these was improperly used to designate the filer for an audit). If, however, decisions are made in an automated fashion by a computer program, subsequent oversight can review exactly the process that led to them, so long as the program records sufficient information about its operation.

Different scholarly communities have suggested very different regimes for the governance of automated processes. Legal scholars often suggest additional transparency or a testing regime as a remedy and fear the inability of human overseers to understand how a computer system actually behaves. Computer Scientists, however, ask for a complete specification of a desired policy *a priori* and then set about designing a computer system that maintains or enforces that policy as an invariant, without much regard for exactly how such a specification comes into being. Below, we describe both approaches and their problems in more detail as well as the project of this

[3]This concern is not entirely hypothetical. In 2013, the United States Internal Revenue Service was accused of targeting certain tax filings for additional scrutiny as a political punishment, the latest in a long history of such accusations. See `https://en.wikipedia.org/wiki/List_of_allegations_of_misuse_of_the_Internal_Revenue_Service` for an enumeration of major instances, including the 2013 scandal.

dissertation: to unify the requirements of both communities and build an interface between their disparate approaches.

Transparency While legal scholars have argued for nearly twenty years that automated processing requires more transparency [94, 95, 97, 121, 293, 294, 327], transparency is not a full solution to the problem of accountability. Full transparency can often be undesirable because it defeats legitimate proprietary trade secrets or permits gaming of a system. Transparency is also insufficient to reveal problems with automated decision making: simply reviewing the code of a computer system is not always enough to know how a decision was made or whether it was made fairly. For example, the output of a program which interacts with its environment in a way that affects the outputs (e.g. by reading from a database, registering the time of day, or determining the load on a system) will not be made reproducible simply by publishing its source code.

Our tax auditing example provides an obvious case for the insufficiency of transparency; we identify three major failings of a transparency-oriented accountability regime.

(i.) A good audit selection process should have a random component, so that even someone with a good model for how audits are meted out cannot cheat with the certainty of not getting caught. And yet, if a program polls its environment for random data (for example, by reading from a system device designed to supply randomness such as `/dev/random` or `/dev/urandom` on UNIX-like systems), it will return different results each time it is executed. Therefore, simply publishing the source code of the program does not on its own provide robust accountability, because it does nothing to ensure that the asserted results of random number generation were fair, and not chosen to reflect a pre-determined result.

(ii.) Transparency defeats legitimate security goals: the goal of a tax auditing system is to increase tax compliance by reducing the incentive to cheat. And yet disclosing the code of a program that selects people for audit inherently allows tax evaders to arrange their filings so as to minimize their likelihood of receiving additional scrutiny.

(iii.) Transparency defeats legitimate privacy goals: a truly transparent system requires that the audit status decisions of individual taxpayers be reviewable, at least to an oversight body. Such a body will of course have to see the filing data of taxpayers whose audit status it reviews, compromising the confidentiality of that information. If decisions must be reviewable publicly, this disclosure problem becomes even more intractable.

Technical Assurances Meanwhile, computer scientists have developed several techniques for ensuring that software systems satisfy well defined predicates. Indeed, this approach could be described as the main project of the enormous subfields of computer security [11] and software verification [12,22,298]. Techniques for providing assurance that an automated process behaves as intended include: (i.) programming language features designed to help programmers write correct programs or detect incorrect ones [298]; (ii.) industry-standard methods for software testing [272]; (iii.) tools for formally verifying software against a specification [12, 22]; (iv.) data analysis methods proven not to violate certain formal fairness guarantees [130]; and (v.) cryptographic protocols designed to enforce particular invariants or to prove that a particular computation was performed [16, 38, 39, 42, 49, 57, 162, 187, 291, 357, 369]. While these well-researched tools are useful and important in building governable automated processes, they all assume that a detailed set of requirements is known in advance which can be specified in precise mathematical language amenable to expression in software code. Computer scientists almost always ask for a specification

that a system is supposed to meet before they can apply one of the above techniques, or at least they assume that such a specification exists in principle. However, the political processes that define oversight and accountability for such systems are rarely equipped to provide such precise requirements. In particular, governance processes often consider the correctness of an operation only after the fact, using details from a particular case to determine if that case was decided correctly. Further, efficient governance often requires some level of compromise in determining specific outcomes, in order to build political support for an action. This vagueness in turn opposes the conversion of laws into software code. It is rare that governance processes are equipped to produce such a complete, detailed, technical specification in advance.

Returning to our tax audit example, we can observe that merely constructing a tax filing correctly is a difficult problem, let alone the problem of evaluating one for fraud risk. In the United States, preparing tax forms is estimated to require billions of person-hours and tens of billions of dollars in specialized services each year.[4] Despite these significant resources, even specialized tax preparation software does not always produce correct filings. Indeed, the process of converting the tax code (or indeed any body of law) into software is laborious and error prone, requiring a large department within the Internal Revenue Service.

We therefore consider traditional computer science approaches to assurance to be complementary to, but different from our approach: we wish to show that results are well-justified, so that auditing and verification processes can function. This is distinct from the important questions of whether the results computed by any particular system are, in fact, the correct results or whether they are the results intended by that system's designers. For a discussion of the relative merits of recording how a decision was reached vs. proving that the decision was correct, see Yee [218]. We

[4] Joshua D. McCaherty, "The Cost of Tax Compliance" *The Tax Policy Blog, The Tax Foundation. 11 September 2014.* `http://taxfoundation.org/blog/cost-tax-compliance`

survey existing techniques for building assurance into computer systems and existing definitions of accountability in the computer science literature in Chapter 2.

Oversight Instead of well defined rules specified in advance, real-world systems typically use *oversight*. Some entity, the *oversight body* or *overseer*, has the authority to demand information from a decision maker and then to hold the decision maker accountable for its actions. The oversight body might be a court, which has the power to interpret the law to say whether the decision maker's actions are lawful, even when the legal status of those actions might have been initially unclear. Alternatively, the oversight body can be a legislature or a higher administrative authority, which can direct a decision maker to change its practices because the oversight body believes those practices are inconsistent with public values or are likely to be found unlawful.

Oversight occurs after the fact (when it occurs at all) and involves complex legal or value judgments. It is an expensive process. We model the oversight body as an oracle that can be consulted only rarely. Decision makers have a model of the oversight body's likely behavior, but this model is not exact, so a decision maker will try to minimize the likelihood of having its actions disapproved by the oversight body, while simultaneously trying to maximize its mission-oriented objectives.

For example, the power to oversee a tax authority's decisions about whom to audit might be vested with a court, a legislative committee, or an internal administrative body within the agency itself. Regardless, that body would have the authority to review decisions on a case-by-case basis, investigating whether the decision to audit or not to audit was consistent with the agency's standards for making that determination. A functionary charged with making audit decisions is therefore driven to only make decisions which he or she believes will be determined on review to have been valid. At the same time, each functionary will apportion the agency's audit resources as best they can to satisfy the agency's goal of maximizing overall tax compliance.

Decisions about whom to audit made by a computer can be audited by reviewing the program that makes them, either by proving that it satisfies particular general properties, such as only selecting taxpayers paying below a certain minimum effective rate or by reviewing specific decisions by considering the input/output relation of the program for a particular taxpayer and determining if that execution comports with the agency's policy for deciding audits.

An interface between technology and the law The challenge explored in this dissertation is how to provide technical assurances which are nonetheless useful in enabling meaningful oversight. That is, we aim to design an interface that enables policy processes to oversee automated decision making systems. This will allow deployments of such systems to be embedded in a governance context in such a way that stakeholders will consider those deployments trustworthy.

Specifically, we propose a novel integration of the technological aspects of program and execution verification with the legal and political oversight process. Our methods are designed to use the tools of computer science to ensure cryptographically the things that can be proven, while providing the necessary affordances for the oversight process to operate effectively. This connects the technical aspects of verified computation with realistic oversight processes. We describe verified computation generally in Chapter 2, Section 2.1.3. Chapter 3 gives an overview of the cryptographic tools necessary to enable our methods and an overview of associated approaches to facilitating accountability and oversight. Chapter 5 describes a cryptographic protocol that defines our proposed interface between formal, technical assurances and the requirements of governance.

As an example of the current state-of-the-art techniques for building accountable systems, which function well when a complete definition of accountability and the properties to be enforced can be specified prior to deployment, we present a novel

cryptographic protocol for compelled access to data by law enforcement which maintains strong accountability in the sense that the guarantees made by the cryptography built into the protocol facilitate robust oversight of what data are accessed, when they are accessed, that they are accessed only by authorized parties, and that parties who are obligated to facilitate that access actually do. We survey the state of the art for designing such systems in Chapter 2. The cryptographic tools we require to build this protocol are developed in Chapter 3; the protocol itself and the accountability properties it provides are presented in Chapter 4.

An often-proposed alternative to our approach is to rely on a trusted expert who is allowed to inspect the system. This can be problematic for several reasons. First, it is often impossible to identify an expert whom all of the affected parties agree to be competent and trustworthy. Second, it can be difficult or impossible to tell whether the program the expert inspected is the same one that was used to make decisions. Third, the authority might not trust the expert to see the internals of the system. Rather than relying on an expert to evaluate the internals of the system, our methods instead use cryptographic proofs to enable traditional oversight processes and to convince a skeptical observer who does not see the full system but only its outputs.

Structure of this Dissertation

This dissertation argues that, no less than when designing a human process or a traditional bureaucracy, those who design computer systems to fulfill important functions—whether in the public sphere or the private sector—must begin with oversight and accountability in mind. We offer a novel example applying state-of-the-art techniques to a well defined accountability problem; survey currently available technological tools that could aid in designing systems to facilitate oversight and account-

ability; introduce a novel cryptographic construction using these tools that facilitates oversight and accountability for a very general class of automated decision systems; demonstrate several examples of how our construction could be applied to real scenarios; and offer suggestions for dealing with the apparent mismatch between policy ambiguity and technical precision.

We summarize the later chapters below.

Chapter 2: Background

Chapter 2 summarizes traditional approaches to oversight and accountability in both computer science and other literatures such as public policy, law, philosophy, sociology, and communications. Specifically, we give a taxonomy for uses of the term "accountability" in computer science, while making our own use of the term precise. Accountability is a widely used term in the computer security literature, but it lacks an agreed-upon definition. We believe that computer systems can only be said to be accountable if they facilitate the goals of the social, political, or legal context into which they are deployed, much as is true for traditional human-mediated systems.

Specifically, we briefly discuss the role and function of law and philosophy surrounding the concept of "rule of law", which undergirds our entire project—we aim to provide tools that enable governance so that end users will trust the actions of decision making systems, much as the rule of law (rather than rule by the whim of an arbitrary ruler) serves to impart trust in governance generally. We also discuss theories about the role of laws and other rules in society [132, 133, 156, 203]. We also summarize recent legal and social science scholarship relating to the governance of computer systems. Much has been written about how to ensure due process [94, 97, 294], why transparency is a useful solution [94, 95, 97, 121, 294, 327], how automated decisions can mask unfair decisions as fair decisions [26, 97, 138, 315], and on the meaning of accountability for automated systems [153, 287].

We also discuss related technologies that can provide technical assurances for automated systems such as software testing [31, 160, 272], type systems [298, 299], systems for software verification [12, 22], protocols for verified computation or cryptographic assurance of a well defined invariant [16, 38, 39, 42, 49, 57, 162, 187, 291, 357, 369], as well as the meaning of "fairness" in computer science [110, 130, 202]. We give a special emphasis to surveying algorithms for fair classification in machine learning. We consider the question of how to certify some definition of fairness in an automated system particularly interesting and important, and return to it in Chapter 6. We also survey the computer science literature and describe systems designed to enable process auditing [11, 109, 194]. We close the chapter with a taxonomy of the uses of the word "accountable" in computer science [146,365], which are many and disparate, and contextualize our definition in terms of enabling the governance of a system in a political, social, or legal context.

Chapter 3: Required Cryptographic Primitives

Chapter 3 defines the required cryptographic tools for the rest of the dissertation, giving both general primitives and specific constructions as appropriate.

In specific, accountable algorithms require cryptographic commitments and non-interactive zero knowledge proofs suitable for verifying executions of general-purpose computations. We construct our implementation using zero-knowledge succinct arguments (zk-SNARKs) [39, 41, 162, 291] and realize commitments in the random oracle model using a hash function on elements of a field based on the hardness of certain lattice problems, due to Ajtai [9, 169]. This choice is advantageous, because Ajtai's lattice-based hash can be represented compactly and computed quickly in a zk-SNARK scheme. Lacking an efficient pseudorandom generator for use in the computation of a zk-SNARK, we elect to generate pseudorandom data outside the zk-SNARK and only verify its correct computation in the proof; further details are

given in Chapter 5. We also survey approaches to achieving verifiably fair and unbiased random values to use as seeds for a pseudorandom generator. Although our construction leaves aside many interesting questions about how to make randomized algorithms accountable, we remark that decision makers fielding our protocol will need to take steps to make it possible to verify that inputs which are supposed to have been chosen at random really were, such as engaging in a separate protocol to generate fair randomness [54], using a trusted random beacon [305], or exhibiting a public ceremony [134].

Our work on accountable compelled data access uses some more specific cryptographic tools, which allow us to leverage the known structure of the problem to build protocols that are much more efficient. The most important of these primitives is identity-based encryption (IBE) [59, 60, 63, 64, 100, 332, 361]. We use the IBE of Boneh and Boyen (BB-IBE) [59, 60] in our protocol, specifically a variant known as blind IBE [182] which is related to oblivious transfer (OT) [183]. We also summarize OT [78, 107, 213, 247, 277–280, 304]. We define in Chapter 4 an OT variant called auditable oblivious transfer to give a useful, compact abstraction defining the invariants provided by our protocol. Chapter 3 also describes secret sharing [17, 32, 51, 71, 82, 92, 119, 163, 266, 296, 331]. Specifically, we require the linear secret sharing scheme of Shamir [331] which we use to define a threshold version of BB-IBE, also constructed in this chapter.

Chapter 4: Accountable Warrant Execution

We have argued that social, political, and legal processes are not equipped to produce sufficiently detailed rules to accommodate the traditional notion in computer science of enforcing a well-defined specification that has been determined in advance. Of course, in at least some cases, it is possible to specify a precise set of requirements in advance and then to enforce those requirements by technical measures. When

this is the case, it is possible (using well studied methods) to build a system that guarantees the desired policy as an *invariant*, proving that it holds either to some set of participating parties, to a designated auditor, or to the public at large. Such computer systems can also be accountable under our definition, but with the caveat that they are less common and are less flexible with respect to changes in policy or the need to handle exceptions and special cases.

We observe that such an approach is similar to the well studied concept of *static analysis* in that the properties of the system in question can be determined from an inspection of the design of the system itself, without the need for interactive testing or runtime observation. However, such an approach may well make use of dynamic methods to establish a certain invariant—what is established statically is the correctness of the dynamic checks themselves and the policy they enforce. Chapter 5, summarized below, describes an alternative approach which can be analogized to dynamic analysis.

As an example of this approach, we offer a protocol for accountable compelled access to data by law enforcement [232]. Law enforcement and intelligence agencies increasingly demand access to data held by providers of communications services,[5] sometimes demanding access to a wide swath of communications metadata such as the details of when and between whom all communications have taken place.[6] Naturally, such compelled access can be cause for concern by citizens at large, and it is clear that while many such accesses are legitimate and necessary, society has a compelling interest in limiting the scope of access to those records which can be adjudged as

[5]See for example a summary of industry-developed tools, referred to as *transparency reports*, for reporting when and how often service providers are responsive to governmental data requests, available at `http://www.forbes.com/sites/kashmirhill/2013/11/14/silicon-valley-data-handover-infographic/`.

[6]As an example, see the leaked order of the Foreign Intelligence Surveillance Court: "Secondary Order, In re Application of the FBI for an Order Requiring the Production of Tangible Things from Verizon Business Network Services, Inc. on Behalf of MCI Communication Services, Inc. d/b/a Verizon Business Services.", available online at `http://s3.documentcloud.org/documents/709012/verizon.pdf`.

plausibly contributing to the legitimate furtherance of an investigation. The power to make this decision, in turn, generally vests in a trustworthy authority such as a court or administrative judge. Specifically, a longstanding requirement for such systems is that access happens only under a legal *order* such as a warrant.

Chapter 4 describes our work on this application scenario, which is joint work with David Wu, Joe Zimmerman, Valeria Nikolaenko, Dan Boneh and Edward W. Felten.[7] This work develops a precise model of the problem under which there is a clear policy to enforce. We describe a cryptographic protocol to provide access control for records subject to compelled access, with the property that any records obtained by the investigator can be provably shown to have previously been authorized by an order from the court. We refer to this protocol as a system for *accountable warrant execution*.[8] Chapter 4 also describes our protocol for enforcing a precise and useful policy under this model; our implementation of this protocol; and extensive benchmarks demonstrating the feasibility of applying this method to realistic deployment scenarios (such as the protection of compelled access to all telephone metadata in a large country).

Chapter 5: Accountable Algorithms

As mentioned, real world systems (including traditional bureaucratic systems) cope with ambiguity as to which policy they should enforce using oversight mechanisms, rendering the traditional approaches to assurances about computer systems, surveyed in Chapter 2, moot. Chapter 5 describes our approach to designing accountable algorithms which, in addition to their ordinary outputs, produce commitments and proofs that can convince the subject of a decision that consistent procedures were

[7]While this work is yet unpublished, a preliminary version has been released as Kroll et al. [232]. The work presented in Chapter 4 represents significant system-level improvements and new experiments over the released version.

[8]This name is somewhat careless, since warrants are only one of many types of orders under which compelled access is possible.

followed. Accountable algorithms also facilitate oversight by a body such as a court, legislature, or administrator that is charged with ensuring that the system's operation is lawful and (in the case of government) consistent with public values. We construct a new kind of tool: rather than enforcing a pre-specified and precise set of requirements, we use cryptography to enable an oversight body to make decisions after the fact about whether outcomes were acceptable or unacceptable. In this way, our protocol serves as an interface between available technical assurances and the realities of political and social processes. This protocol is joint work with Edward W. Felten, to whom is due the idea of using techniques from verified computation to enable oversight.

Critically, our protocol can ensure the properties we desire while maintaining the secrecy of some information from some parties, limiting disclosure to precisely the information necessary to achieve accountability but no more. To take our earlier tax audit example, it is ideal to prevent an overseer from learning anything about the filing data of individual taxpayers, but (at a minimum) certain specifics must be disclosed in order to prove any misbehavior on the part of the tax authority.

Our protocol also provides accountability for randomized decision processes, such as lotteries or other rules that apportion scarce resources by chance. Specifically, we require that any random coins required by some execution of a program be replaced with pseudorandom coins generated from a suitably random seed given as distinguished input and produced by a process for generating fair randomness. Many such tools exist—we survey the options in Chapter 2.

As a concrete example, consider again a tax collection authority tasked with deciding which taxpayers to audit for tax compliance. The authority wishes for taxpayers to trust its enforcement choices but cannot simply publish its selection rules for fear that this will enable tax evaders to cheat with confidence by maximizing their likelihood of evading auditing and enforcement. Even if the rule is randomized such that a published rule does not guarantee freedom from scrutiny, taxpayers have no way

of knowing that they were not in fact selected for improper reasons, with a falsified random selection covering the agency's malfeasance. Our protocol allows the agency's decision process to be cryptographically bound to the agency's result, enabling strong after-the-fact oversight of individual decisions.

Specifically, we achieve accountable algorithms by describing a new cryptographic protocol parametrized over a *cryptographic commitment scheme* and a general purpose *zero-knowledge proof system* suitable for applications requiring *verified computation*. We define these primitives in their general form in Chapter 3 and also describe efficient realizations of them for use in our protocol. In Chapter 5, we describe our protocol for constructing accountable algorithms as well as our implementation of this protocol and its performance in general. Chapter 6 describes some specific example scenarios in detail and the required overhead of addressing these scenarios with our protocol.

Continuing the analogy from the previous chapter summary, this approach can be thought of as a kind of *dynamic analysis* in that the correctness of a system with respect to a policy is determined from observation of the system in action. Our approach differs from traditional dynamic checks, however, in that all actual checking is deferred to a later oversight period. Prior to oversight, during actual operation, the system merely certifies that this deferral is valid because it knows the requisite inputs for the check (namely, the input/output relation of some function, including any secret inputs and random coins). This is, however, not a certification or a guarantee that the check made during the oversight period will pass—it is entirely possible that the oversight body will later disapprove of the system's actions and take action to remedy or punish any perceived misbehavior.

Chapter 6: Example Accountable Algorithms

We demonstrate the practicality of our techniques in Chapter 6, giving a basic performance evaluation of the protocol from Chapter 5 and describing several example

scenarios which would benefit from accountable algorithms. For each scenario, we provide an example implementation and describe its performance. Finally, we discuss some preliminary work on designing useful programming abstractions for accountable algorithms and for zk-SNARK-based systems in general.

Specifically, we review a basic classification scenario, in which a linear model is used to assign a score (such as a credit or other risk score) to a large number of people. Such models can range from simple inner products with a secret set of trained weights, which are straightforward to represent compactly in the arithmetic circuit form required in our implementation using zk-SNARKs, to complex kernel-method classifiers that must be approximated in order to achieve reasonable performance in a zk-SNARK. Such models are also a useful starting point for an investigation of our methods for classification tasks, which often rely on linear models

We also consider classifiers with formal fairness properties, such as those investigated by Hardt [202] and Dwork et al. [130]. Such classifiers provide the opportunity to explore an exciting extension to the protocol of Chapter 5 which allows us to prove that an invariant holds over a computation without showing a complete execution trace of that computation.

Lastly, we consider accountability for the U.S. State Department's Diversity Visa Lottery, a program with totally transparent rules specified in statute, yet with a troubled history of accountability. Specifically, in the 2012 fiscal year, the State Department was forced to rescind its initial computation of lottery winners because the computation of the winners had been performed incorrectly. Further, because the nature of the computation is opaque to the lottery entrants, many fraudulent services prosper by selling entrants services to "expedite applications" or "improve their chances". Finally, the opacity of the lottery process can mean that winners or their families can be viewed as receiving a reward for collaborating with U.S. interests, and so may be subject to reprisals. Accountable algorithms cannot guarantee that

the lottery is performed correctly, although they can prove that fair randomness was used in a fair lottery and provide evidence as to what failed if such a process were to go wrong.

Chapter 7: Conclusion: Designing Computer Systems for Oversight

We conclude in Chapter 7 by describing how to design computer systems so that they admit robust oversight and accountability—both for the straightforward problem of ensuring procedural regularity and the less straightforward problem of ensuring some sort of fairness, anti-discrimination, or legal compliance property—using our protocols and other techniques from computer science. This chapter is based on a paper workshopped at the 2015 Privacy Law Scholars Conference, which is joint work with Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu [233].

Specifically, we give recommendations for policymakers about how to incorporate these techniques into requirements on those designing and fielding consequential automated decision making systems. We also give actionable recommendations for computer scientists researching future accountability technologies. Ultimately, as computer-mediated decision systems become more common, such requirements will be necessary to integrate these decisions into traditional political processes and therefore to achieve legitimacy for automated decision systems.

This chapter also includes a discussion of the role of technical tools in achieving social ends, in which we carefully engage an argument of Narayanan [281, 282] about the non-usefulness of cryptography for achieving socially defined goals such as privacy. We attempt to explain why our approach sidesteps the pitfalls in deploying cryptographic systems to situations with poorly defined goals identified by Narayanan.

Chapter 2

Background

This chapter presents the context for accountable algorithms as well as a survey of the related literature in computer science as well as in other fields such as law, public policy, sociology, philosophy, and communications. This chapter covers contextual concepts—related cryptography and the specific cryptographic primitives we rely on are described in Chapter 3. Reflecting the substantial differences between how these issues are framed in computer science and other fields, we separate our discussion of the extensive related literature at the boundary of computer science and other fields, summarizing related concepts from computer science in Section 2.1 and related concepts from other fields in Section 2.2. We describe a taxonomy of uses of the term "accountability" in computer science in Section 2.1.8, define our own use of the term in Section 2.1.9, and describe its meaning in other fields in Section 2.2.7.

2.1 Related Concepts from Computer Science

We begin by considering related ideas from computer science. Our review casts a wide net, covering a broad swath of technologies and techniques that can improve confidence that a system is behaving in the desired way.

2.1.1 Software Testing

Software testing is a structured investigation by the developer of a computer system to assess the quality of that computer system and its functionality. Testing is distinct from unstructured review of a system's operation and from "debugging", or responding to problems as they occur. Myers gives a more careful treatment of this distinction [272] and surveys software testing methodology. Debugging, however, is still often integral to the development process. Gelperin and Hetzel [160] give a history of approaches to and goals of verification methods used in the software development industry. Once considered unnecessary, software testing later became a discipline unto itself within the software industry separate from development, commonly performed in a separate department by specialized engineers. More recently, state-of-the-art software development methodologies have advocated for testing to become an integral part of development, performed by the original development team as software is written. Some advocates even suggest that testing should precede development, a model known as *test-driven development*, as it helps sharpen in the developer's mind what a program should do [31].

Tests may be separated into broad categories using two major distinctions: the distinction between *static* and *dynamic* testing; and the distinction between *black-box* and *white-box* testing.

Static testing operates on software source code itself and consists of activities such as code reviews, code walkthroughs, and code inspections. It is essentially a form of proofreading, although it may be undertaken by more than one person or performed in a structured way. By contrast, dynamic testing involves running the program and examining its behavior when given certain inputs. It may be performed at many levels of abstraction, from evaluating the behavior of an entire program (sometimes called *end-to-end* tests or *integration* tests) to examining only the execution of a single function or statement (so-called *unit* tests). Often, dynamic testing methods

are supported by tools such as stub or mock functions, which replace parts of the program which are incomplete, unpredictable, or impractical to use when testing; or debuggers, which allow execution of the program in a controlled environment which may be carefully examined or altered during execution for testing purposes.

Testing methods may be further divided by whether the tester has access to the source code and design of the software system under investigation. If access is possible, testing occurs in a white-box environment; if not, only black-box testing is possible. Most static testing is naturally white-box, as it involves reviewing the code itself; however, black-box static testing is possible in the form of testing a specification or requirements document. Black-box dynamic testing can only observe the input/output relation of a program, and so inherently cannot know whether the program will behave very differently on inputs not already tested. White-box dynamic testing, however, can use knowledge of the program's structure to find classes of inputs for which exemplars may be tested and results safely extrapolated to the entire class. Such analysis can also be used to measure *test coverage*, or an approximation of how much of a program's behavior has been reviewed by testing. More advanced white-box methods such as *fault-injection* testing and *mutation* testing modify the program or its environment either as it is running or between test runs to determine how it responds to inconsistent or hostile data or to improve the set of available tests.

Because testing all inputs is combinatorially hard, testing for security properties often involves feeding a program randomly chosen inputs which may not have any semantic meaning, in an effort to find unexpected misbehavior in a program. In a white-box setting, such testing may be greatly improved by some simple program analysis, as investigated by Godefroid, Levin, and Molnar [165].

So-called *grey-box* testing uses knowledge of a program's structure and function to create test cases, but then executes those cases from a black-box perspective. Accountable algorithms enable a sort of grey-box testing, in which an oversight body

can learn the structure and design of a process and determine what information ought to be made public about each decision in order to engender general acceptance of the overall computer system.

2.1.2 Type Systems and Software Verification

In nearly all programming languages, there is a set of rules that assigns a property known as a *type* to various syntactic constructs such as variables, expressions, statements, functions, or modules. The type of a programming construct is based on what kind of value it computes; the purpose of type systems is to define and annotate interfaces between different parts of a program so as to prove the absence of undesirable behaviors by checking different parts of the program for consistency with one another. This checking may happen at compile time (that is, statically), at run time (dynamically), or Both. The theory of types is large and rich, but at its core aims to ensure that programs have a well-defined meaning. Advanced type systems can encode complex correctness properties about the function of a program, and programs written in languages with such type systems may be more amenable to formal verification. An excellent overview of the theory and use of types may be found in Pierce [298, 299]. An example of a complete system for producing high-assurance software with proofs of correct operation in all cases is due to Appel [12].

Types form one approach to the classical problem of *formal software verification*, the study of whether a program meets its specified requirements. Formal methods aim to prove in the tightest way possible the correspondence between the execution of a piece of software and a given specification for that software. Beyond encoding the specification in the type of the program or its components, formal methods also encompass various types of *program synthesis*, or the automated derivation of executable software from a specification; *model checking*, or the exhaustive investigation of all configurations of a finite-state system; manual and automated *theorem proving*,

or building machine-checkable assertions that a program satisfies a certain desirable property; and *semantic analysis*, or the formal assignment of meaning to phrases in a programming language in one of many simple computational models.

While formal methods are not widely used in industrial software development, they have long shown promise and do see deployment in areas with hard requirements for few software bugs, such as aviation, vehicle safety, and systems for handling highly classified information. In particular, model checking has been used to substantial benefit in the hardware industry to verify that complex chips, which are developed in ways very similar to large software systems, do not have unintentional bugs. An excellent overview of model checking may be found in Baier [22].

Formal methods have also been applied to verify the correctness and accountability of data use by automated decision systems. Tschantz and Wing give a methodology [351], while Tschantz et al. give more detailed experiments with real systems, including a verification that certain systems satisfy the stringent requirements of differential privacy [348–350].

Formal methods of all stripes are complementary to accountable algorithms. Our techniques make it possible for an oversight body or the general public to determine *what* happened in an automated decision, but not that this action was the correct action (i.e., the one intended by the system's designer). An accountable algorithm can still be buggy or perform outside the intentions and expectations of its designers. However, we remark that the process of introducing the tools necessary for accountability and oversight requires a careful review of how a process works, which may in turn expose bugs and demonstrate areas for investigation by way of testing or formal verification. We discuss the value of reviewing processes for accountability further in Chapter 7. We observe, however, that formal methods are very much about relating the behavior of a specific implementation of a program to its specification. Therefore,

they are not directly applicable in the setting we consider, where specifications are subject to review and oversight.

2.1.3 Verified Computation

Verified computing addresses the problem of outsourcing the computation of a function to an untrusted worker, such as a large server farm or a distributed set of clients. Two common scenarios motivate the problem of verified computing: first, a weak client such as a mobile telephone or single laptop may wish to outsource a very large computation to a cloud computing provider while guaranteeing that the provider actually performs the computation, since the weak client cannot check this directly; second, a large data processing project such as U.C. Berkeley's Search for Extraterrestrial Intelligence At Home ("SETI@home", one of the largest distributed computing projects ever undertaken) may wish to break a large computational problem into small pieces to be given to untrusted clients for computation without allowing clients to return plausible results without actually performing the computation. In the second scenario, the provider of computational work has the added advantages of being able to compute the work itself and being able to (at the cost of some loss in efficiency) send a sample of already performed work out for reprocessing to check the returned results for consistency.

A classical solution to convincing a weak verifier of the truth of an assertion it could not compute itself comes from the theory of interactive proofs [175], described in slightly more detail in Chapter 3, Section 3.1.2. In particular, work on interactive proofs led to the development of *probabilistically checkable proofs* [16], in which a prover can prepare a proof that the verifier need only check in a very small number of places (the proofs for $\mathcal{NP}$ languages require only a constant number of bits). However, the entire PCP proof object may be very long—perhaps too long for the verifier to process. Various improvements to PCPs have been proposed, ultimately leading to

the work on succinct argument systems presented in Chapter 3, Section 3.3.2 and used in this dissertation for the protocol of Chapter 5. A more complete history of work in this area, as well as a formalization of the verifiable computation problem, may be found in Gennaro, Gentry, and Parno [161].

2.1.4 Fairness in Computer Science

The term "fair" in computer science has been used to describe very simple properties, such as properties related to the sharing of resources, and also much more difficult properties, such as properties of classifiers that imply similar treatment for test points with certain attributes. We briefly survey both kinds of uses here.

Fair Scheduling and Resource Allocation

The simplest notion of "fair" in computer science is that a number of processes competing for a resource (disk, memory, CPU time) receive equal shares (in expectation, or within some asymptotic bound) of that resource according to the amount they requested or perhaps according to some higher level policy such as equal shares per user, which are then subdivided per process. For example, on a system with four users each running a single process, each process should receive 25% of the CPU time. If one user starts a second process, that user will receive 25% of the CPU time overall, but each of her processes will only receive 12.5% individually. A higher level of abstraction places users into groups and apportions shares of a resource to each group. Shares may also be weighted, to apportion extra resources to important tasks or groups.

Fair resource scheduling is an instance of the well-studied fair queuing problem [184, 337], itself a question about how to assign priorities in a priority queue [8,

102]. A full survey of this literature is beyond the scope of this chapter. Fair schedulers are widely deployed in modern computer systems.[1]

Finding algorithms for fair division of resources is a central problem of 20th-century game theory, although the problems it addresses are very old indeed, dating at least two thousand years [18]. The exemplary problem in this area is the problem of splitting a piece of cake between two players in a way that is *envy-free* (i.e., the players are both happy with the outcome; neither would prefer the other's piece of the cake) [19], a problem and solution described even in the Bible. This problem is closely tied to the general technique in cryptographic protocols of *cut and choose*, in which one player must commit to the partition of some set of secret values and another verifies the validity of this partition by examining one or more elements for which the underlying secrets are revealed. These techniques have been expanded in the game theory literature to cover situations with multiple players, disparate utility among the players (e.g., if half the cake is of a different flavor from the other half, how do the players apportion pieces according to their utility of flavors), and multiple levels of resources (e.g., dividing rent in a house with as many bedrooms as housemates). A summary of this vast literature can be found in Brams and Taylor [68,69] or Robertson and Webb [314].

Unbounded Nondeterminism

A generalization of fair resource allocation is the notion of fairness in systems with unbounded nondeterminism, defined as systems of concurrent processes where the amount of delay in servicing a request by one process can become unbounded. Dijkstra argued that such systems are impossible in practice [124], leading Hoare to argue that

[1]For example, the popular open-source operating system Linux implemented the first fair-queuing-based process scheduler in a general-purpose modern operating system, the "Completely Fair Scheduler" by Ingo Molnár, based on a priority queue implemented as a red-black tree. Prior to this, fair scheduling had only been applied to network queues in desktop operating systems. See `http://lwn.net/Articles/230501/`.

efficient systems should strive to maximize *fairness* [207]. Hoare defined fairness as the property that if the set of processes enters a particular state infinitely often, it must take every outgoing state transition from that state. Equivalently, every possible state transition must occur in an infinite computation.

The halting problem [352] severely constrains the utility of models of computation with unbounded nondeterminism—the halting problem can be solved using the fairness constraint in at least some models of computation with unbounded nondeterminism [339]. The very powerful model of nondeterministic Turing machines only exhibit bounded nondeterminism. This is one of the fundamental barriers to robust formal reasoning about concurrent computations. To resolve this, Hewitt introduced the Actor model [206], later formalized by Clinger [99], which defines sets of concurrent processes that can each be computed by a Turing machine, even though the operation of the entire set cannot. The actor model, along with Hoare's model of communicating sequential processes (CSP) [207], are important tools for reasoning formally about networked systems, which can easily exhibit unbounded nondeterminism (e.g., if a network link fails).

Fair Classification

A more complex notion of fairness arises from classification problems in machine learning. This is relevant to automated decision processes, which often rely on learned models. Section 2.2.6 discusses practical failings of such systems as reviewed in the social science and legal literature. Section 2.1.7 discusses efforts in computer science to examine the fairness of complex automated decision making systems in practice through large-scale measurement and reverse engineering of models. This section

describes the state of the art in machine learning systems which satisfy well-defined formal fairness properties.[2]

One commonly understood way to demonstrate that a decision process is independent from sensitive attributes is to preclude the use of those sensitive attributes from consideration. For example, race, gender, and income may be excluded from a decision-making process to assert that the process is "race-blind", "gender-blind", or "income-blind." 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 implicitly encoded in, or highly correlated with, other non-excluded attributes. For example, when race is excluded as a valid criterion for a credit decision, illegal redlining may occur when a zip code or street name is used as proxy that closely aligns with race.

It seems clear that this type of input "blindness" is insufficient to comport with intuitions of fairness. Although there are many conceptions of what fairness means, 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 et al. [130]. This work stems from a longer line of research on mechanisms for data privacy [128, 129]. We describe further 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. For example, if an apartment complex requires renters to maintain a certain minimum income to hold an apart-

[2]Some material in this section is drawn from a paper workshopped at the 2015 Privacy Law Scholars Conference which is joint work with Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu [233].

ment, then anyone who meets the income threshold can lease an apartment, and so the process will be fair for individuals. However, if the distribution of incomes is very disparate across subgroups of the population (e.g., if a particular racial minority has a significantly lower income distribution), then the process will tend to exclude some groups. As we shall see below, it is sometimes the case that a sufficiently restrictive notion of individual fairness also implies certain definitions of fairness for groups.

Although it is almost certainly the more policy-salient concept, group fairness is more difficult to define and achieve. The most commonly studied notion of group fairness is *statistical parity*, which says that an equal fraction of each group should receive each possible outcome. While statistical parity seems like a desirable property 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. A malicious advertiser wishing to exclude a minority group from the 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 there are minority candidate applications, that is no guarantee that the number of people hired will reflect the distribution either of applicants or of the general population. And the company could hide discriminatory practices by inviting only unqualified members of the minority group, effectively creating a self-fulfilling prophecy as described in Barocas and Selbst [26].

The work of Dwork et al. identifies an additional interesting problem with the typical "fairness through blindness" approach: by remaining blind to sensitive attributes, a classification rule can in fact select exactly the opposite subset of a protected status group from the subset intended! 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 name. 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, and so it is common for people in these cultures to have names which are not shared with anyone else. However, because a majority of accounts come from the majority of the population, for which unique names are rare, any classification based on the uniqueness of names will inherently give a higher false-positive rate for real minority profiles than for majority-group profiles, and may also have a correspondingly high false-negative rate. This unfairness could be remedied if the system could recognize and act on the minority status of a name under consideration, since then the system 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.

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. Specifically, Dwork et al. require that this difference in chance of outcome be less than the difference between individuals subject to classification, drawn from a set V. This amounts to a Lipschitz condition on the classification function under a similarity metric for individuals, $d : V \times V \longrightarrow \mathbb{R}$ meant to be defined per-task. This requires: (1) a mathematically precise notion of how similar or different people are, which might be a score of some kind or might naturally arise from the data in question

(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); and (2) that this notion of similarity capture all relevant features, including possibly sensitive or protective attributes such as minority status, gender, or medical history. Given such a task-specific metric, Dwork et al. define a classification M as a function $M : V \longrightarrow \Delta(A)$, where A is the set of possible classification outcomes. That is, a classifier is a function from individuals to distributions over outcomes. Letting D be a measure of distance between distributions, the Lipschitz condition is:

$$D(Mx, My) \leq d(x, y) \qquad \forall x, y \in V$$

Because this approach requires the collection and explicit use of sensitive attributes, the work describes its definition of fairness as "fairness through awareness". 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 [150, 155, 222, 374].

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 (e.g., at a minimum, x members of a certain minority group should be interviewed for a job) should have a minimal impact on the general fairness principle that similar people are treated similarly. Specifically, the work of Dwork et al. describes a linear program which maximizes overall fairness, enforces the affirmative action requirement, and minimizes loss of classification utility under the Lipschitz condition described above. 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 decisions satisfy 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, care must be taken to avoid biasing the training process in some other way or overfitting the model to the non-representative 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 et al. [374]. 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. Specifically, given a set of users X out of a universe U and a random variable S which defines whether a user is protected, we let $X^+ \subset X$, $X^- \subset X$ be the subsets of X for which $S = 1$ and $S = 0$, respectively. Then our goal is to learn the multinomial random variable Z, which takes K possible values, to each of which is associated a prototype $v_k \in U$. The fairness requirement is that

$$P(Z = k|x^+ \in X^+) = P(Z = k|x^- \in X^-) \quad \forall k$$

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. 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 and generate fair results. However, this approach relies by construction on the problematic definition of group fairness as statistical parity.

A related approach is to use a technique from machine learning called regularization, which involves introducing new information to make trained models more generalizable (usually in the form of a penalty assigned to 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. Examples of this technique is the work of Kamishima, Akaho, and Sakuma [223], the works of Hajian et al. [195–198] and the work of Zafar et al. [373].

The approach 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, computationally indistinguishable random data such that no classifier can tell whether a randomly chosen input was drawn from the real data or the fake data. And further, 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 generated at random, this technique is useful when training a classifier on real data would create privacy concerns. Also, synthetic data can be made public for others to use, although care must be taken to avoid allowing others to infer facts about the underlying real data. Such model inversion attacks have been demonstrated in practice (e.g., by Fredrikson et al. [152],[3]), along

[3]Although Fredrikson et al. conclude that their attack means that differential privacy presents an insurmountable choice when considered as a privacy vs. utility trade-off, their attack says much less about the utility of differential privacy than they claim—Fredrickson et al. determine that a patient's cardiac health status is holographically encoded in their dose of Warfarin, a blood thinner, and therefore that robust techniques to obscure one reduce the utility of a model at producing the other. In fact, this is not an argument against differential privacy, but rather for training a model that both preserves privacy and predicts effectively. The authors did not investigate whether a more robust fair modeling technique could have bridged this gap.

with other inference or deanonymization attacks that allow sophisticated conclusions without direct access to the data those conclusions are about. Such attacks are summarized by Narayanan and Felten [283] and Narayanan, Huey, and Felten [284]. All of these approaches demonstrate that it is possible to build a wide variety 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. While no bright-line rules exist that would allow the designer or operator of such a system to guarantee that their behavior is compliant with anti- discrimination law, it is certainly not the case that no options exist or that unconstrained use of data analysis is necessary or even defensible.

Many of these approaches rely on the insufficient notion of group fairness by statistical parity. One way more 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 law and 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 are best used in specific real-world applications. Accountable algorithms can help bridge this gap by allowing partial model transparency and by enabling certain design decisions to be deferred and reviewed by an overseer.

A complementary notion to machine learning systems that can guarantee pre-specified formal fairness properties is the work of Rudin on machine learning systems that are interpretable [321]. Such systems generate models that can be used to classify individuals, but also explanations for why those 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 domain experts and overseers, who

wish to avoid treating models as black boxes. Again, accountable algorithms complement such systems well by allowing partial transparency, oversight, and verification for the interpreted classifications in individual cases.

Also related is the concept of the security of machine learning systems, or the question of how robust such systems are against manipulation by interested parties. For example, a system that classifies e-mail as spam vs. not spam should have the property that a spammer should not be able to find an e-mail efficiently which both passes the filter and is effective for the spammer. Lowd and Meek [253] investigate the problem of quantifying the difficulty for an adversary to manipulate or game a machine-learning-driven decision system, which they call *adversarial learning*. Barreno et al. [27] give a broader overview of the security of machine learning systems generally, showing that the process of making inferential predictions can be gamed and may leak information about the underlying model or training data inadvertently. These questions of robustness relate strongly to the question of how to interpret the meaning of model outputs, including classification confidence levels: it has been shown in several cases that models can easily be fooled by random test vectors, for example, leading to the practice in activist communities of investigating "dazzle" makeup to fool face recognition systems [317].

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 a simple Bayesian argument: if the fact that an individual was denied a loan from a particular bank tells you that this individual is more likely than you knew before 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 kind of 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 of the individual to be let alone" in that generally people are less concerned with the act of disclosure for their private information but rather with the idea that disclosure interrupts their enjoyment of an "inviolate personality" [360].

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", which Solove argues can be countered by giving subjects knowledge of and control over their own data. It is hardly a wonder, in this framework, that the predictive models of automated systems, which might use seemingly innocuous or natural behaviors as inputs, would create anxiety on the part of data subjects [338].

We can draw an analogy between data analysis and classification problems and the more familiar data aggregation and querying problems which are much discussed in the privacy literature if we consider decisions about an individual as representing (potentially private) information about that individual. 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 a data analysis fair by removing protected attributes from the input data. However, both approaches fail to provide their promised protections, as summarized by Narayanan and Felten [283] and Narayanan, Huey, and Felten [284] as well as by Ohm [289]. 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.

The work of Hardt [130, 202] relates the work on fairness by Dwork et al. to work on differential privacy by Dwork [128, 129]. As differential privacy is a well-founded notion of protection against inferences and the recovery of an individual identity from "anonymous" data, so is fairness through awareness a sound notion of fairness for individuals and a theoretical framework on which to ground more complicated notions of fairness for protected groups. Dwork and Mulligan [131] relate fairness and privacy in a legal context, informed by the computer science techniques described in this section.

The many techniques of building fair data analysis and classification systems described in this section mostly have the property that they require decision makers 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 has led to a policy regime where decision makers are explicitly barred from holding such information, such that deployment of these technical tools would require a policy change.[4] The techniques described in Chapter 5 could be used to

[4]One example is the privacy regime created by the Health Insurance Portability and Accountability Act ("HIPAA"–110 Stat. 1936), 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.

make such a change less prone to engendering the very real concerns of data abuses that have led to the current regime.

A summary of work on fair classification and its relationship to privacy can be found in the anthology *Discrimination and Privacy in the Information Society: Data Mining and Profiling in Large Databases* [110].

2.1.5 Systems for Cryptographic Assurance

The most common approach in the literature to assuring some property to an outside party is to use cryptography to enforce a well defined specification and to give partial transparency into a process to outsiders. Our methods are very similar to this approach, although we focus on enabling oversight so that the invariants enforced by a system do not have to be decided in advance. Many such systems are described in standard references such as Anderson [11].

A general approach to using cryptography to build secure protocols that balance the disparate goals of mutually distrustful parties comes from Yao's protocol for secure two-party evaluation of arbitrary functions [369]. This technique has been improved, generalized, formalized in different contexts, and made more efficient by many authors to create a robust literature on *secure multiparty computation* [37, 81, 86, 91, 180, 214, 230, 231, 247, 257, 300, 334, 357]. Yao's protocol even saw real-world deployment, if only for a rather obscure sort of auction clearing for sugar beets in Denmark [57]. An alternative general method for secure multiparty protocols is due to Goldwasser, Micali, and Widgerson [177].

Another general approach comes from the idea of *fully homomorphic encryption*, or encryption that commutes with a sufficiently expressive set of operations that one can construct meaningful computations on encrypted data. While the promise of such technology was recognized at least as early as 1978 by Rivest, Adleman, and Dertouzos [312], and encryption systems which allowed limited computation on

encrypted data have been known since at least that time (the basic algorithm of Rivest, Shamir, and Adleman is homomorphic for multiplication) of Goldwasser and Micali the mid-1980s and others through the late 80s and 1990s [43,111,174,290,313], no plausible constructions for fully homomorphic encryption were known until Gentry introduced a candidate construction in 2009 [164] and Van Dijk et al. introduced a second candidate in 2010 [355].

A further exciting breakthrough in this area comes with the release of a candidate construction for *indistinguishability obfuscation* (IO), a notion due to Barak et al. [25], from to Garg et al. [158]. IO can be viewed as a sort of dual problem to fully homomorphic encryption, in which the computation is encrypted instead of the data. Specifically, IO says that, given two circuits C_1 and C_2 with the same input/output relation but a different internal implementation, there is no probabilistic polynomial time adversary $\mathcal{A}$ that can distinguish between obfuscated versions of the two circuits $\mathsf{Obf}(C_1)$ and $\mathsf{Obf}(C_2)$. Sahai and Waters [323] and later Boneh, Sahai, and Waters [65] introduced the related concept of *functional encryption*, which is a technique similar to identity-based encryption (See Chapter 3, Section 3.2.1), but where the extracted key allows decryption of some function of the plaintext. Goldwasser et al. [172, 173] showed that IO implies succinct functional encryption for all circuits. However, all current constructions are implausibly inefficient.

A brief review of the recent systems security literature yields several systems that fit the model of cryptographically ensuring an *ex ante* specification [116,137,149,188, 234, 260, 297, 346, 354, 359, 363, 376].

A prime example of using cryptography to guarantee certain socially important properties is the plethora of techniques for cryptographically enforced *end-to-end secure voting schemes* citekarlof2005cryptographic, adida2006advances, adida2006scratch, adida2008helios, ryan2009pret, gritzalis2012secure. In particular, systems such as Adida and Rivest's Scratch and Vote [7], Adida's Helios [6] and Ryan

et al.'s Prêt à Voter use cryptography and other tools to guarantee to voters that their ballots satisfy certain invariants such as *ballot casting assurance*, the property that each voter can learn that their ballot was correctly recorded and that the total election count corresponds to the cast ballots. Many such systems and their specific properties are surveyed in Adida [5] and Gritzalis [185].

Another common use of cryptographic assurance for particular invariants is to guarantee that some process prevents certain types of information flow. These systems are often confusingly called systems for *privacy-preserving* computation, when in fact they only capture privacy in terms of preventing the flow of information. However, as we describe in Section 2.1.4, privacy can encompass harder-to-formalize properties such as fairness. We discuss yet further subtle properties of privacy throughout this chapter. The relationship between privacy and accountability is explored some in Section 2.1.8. In Chapter 7, we explore this relationship in the context of deploying cryptography for social assurances and relate our approach to accountability to prior efforts to use cryptography to guarantee privacy.

One well developed line of work in this area is the line of privacy-preserving advertisement delivery systems [189, 344]. Such systems aim to allow the targeting of users for advertisement purposes (in the sense of serving users who visit sites in certain categories advertisements tagged by those categories) without revealing to the advertiser or even the advertising network anything about the browsing behavior of particular users. Two major systems, Adnostic by Toubiana et al. [344] and Privad by Guha, Cheng, and Francis [189] bear specific mention and spawned a deep literature of closely related protocols.

Another line of work in this area relates to the specific problem of collecting tolls from drivers electronically without revealing to the tolling agency the location track of any specific vehicle [23, 259, 301]. VPriv, by Popa, Balakrishnan, and Blumberg [301] further considers the question of how to aggregate tracks so as to learn summary

statistics about where they start and end. Systems such as PrETP by Balasch et al. [23] give a more complete and practical solution for electronic tolling specifically, while Meiklejohn et al.'s work considers how to preserve the privacy and security guarantees (e.g., that tolls paid actually represent the tracks driven) of such systems when drivers are allowed to collude [259].

A problem with the deployment of such systems is that the invariants they preserve must reflect the realities of the environment to which they're deployed. An example of this mismatch is the "breaking the glass" problem in medical records systems [147], which requires that medical personnel be able to override access controls in an emergency situation, something that is at least difficult to achieve if record access is controlled cryptographically. We consider the relationship between cryptographic models of assurance and real-world social requirements further in Chapter 7.

2.1.6 Auditing Systems

The use of *auditing* and *audit trails* to verify the behavior of a system has an extensive literature in computer science and beyond. We give a necessarily incomplete overview of methods and application scenarios here. When audit trails or logs are kept, it is important to take steps to ensure that they cannot later be modified to mask any malfeasance. Crosby and Wallach give a technique for building cryptographically assured tamper-evident logs [109]. Haeberlen, Kouznetsov and Druschel give a method for detecting and rectifying misbehavior in distributed systems [194].

The technique of audits comes from the world of business financial controls, where it still plays a large role. Anderson gives an excellent summary of the uses of audit trails and the risks of misuse or over-reliance [11, Ch. 10]. The basic theme of financial audit trails is the concept of the *double entry ledger*, which is a two-part record of transactions processed by a system, where the parts are meant to be kept under separate administrative control. By always offsetting an entry in one part of

the record (say, a credit) with a corresponding entry in the other (say, a debit of an equal amount), maintaining the invariant that the books *balance*, such that the sum total of all transactions in both parts is always zero. Separating concerns in this way controls abuse, since a malefactor must maintain the balancing invariant and must therefore forge two entries in distinct audit trails, which is presumably harder than forging only one. Double entry bookkeeping has existed in commercial use since at least the 14th century [240].

Auditing is required by several compliance regimes enshrined in law, such as the Sarbanes Oxley Act[5] and the Health Insurance Portability and Accountability Act.[6] Such audits are often conducted by large accounting firms under a variety of different standards for risk management. A survey of these standards is given by Colbert and Bowen [101]. The goal of such audits is to define risks to an organization or process and determine a formal owner for each, as well as to identify the proper means of control or mitigation for organizational risks. Anderson also describes this process and its risks [11, Ch. 25]. Blocki et al. give an game-theoretic model of compliance as it relates to the likelihood of detection by an auditor for systems using artificial intelligence [53]. We discuss compliance regimes further in Section 2.2.5.

Systems that enforce privacy requirements on large databases perform what is referred to as *query auditing* or sometimes *query overlap control* to determine if a query or its response will be unduly intrusive or reveal too much about the database. Some techniques track what the querying party has already learned to determine if additional disclosure will reveal a sensitive statistic. Many schemes exist for automating such analysis, which often bears resemblance to systems for information flow and access control in operating systems, but which can be more inferential in nature [90, 126, 225, 274]. Adam and Worthmann give the classic survey of these techniques [4]. Nabar et al. give a more up-to-date survey [273].

[5]Pub.L. 107–204; 116 Stat. 745.
[6]Pub.L. 104–191; 110 Stat. 1936.

Audits have long been a standard part of the operation of voting systems and the approach taken by such systems to ensuring that results are verifiable and trustworthy to voters, candidates, and sitting governments. Audits may take many forms: if votes are recorded in multiple ways, such as electronically and separately on paper, these records can be checked for consistency. If only one record exists, subsets of ballots from similar geographies or from within particular precincts may be compared for statistical similarity. Modern techniques advocate assigning a confidence level to the outcome of an election through an audit, much as many measurements in science are assigned confidence values based on the robustness of the equipment used and the expected repeatability of the measurement. This technique is known by the confusing term *risk-limiting auditing*. Hall gives an excellent bibliography of election auditing literature, which we do not repeat here [199].

Intrusion detection systems can be seen as a form of auditing: they review network traffic and logs to discover patterns of misbehavior that are indicative of attacks and exceptional access. Indeed, early work on intrusion detection was referred to in the literature as *audit trail analysis*; Lunt gives a survey [254]. A classic formal model for intrusion detection analysis is due to Denning [117]. Bishop presents a standardized format for audit trail information to improve analytic capability [48]. Other authors provide systems for performing automated analysis of such information [191, 204, 255, 316].

We draw a distinction between auditing as it is traditionally conceptualized in computer science and accountability as we define it for purposes of this dissertation. Specifically, auditing is a technique that protects the *operator* of a system, and does little to increase the trust in that system by *users* or *subjects* of decisions made by that system. This is due to the fundamental fact that a user or subject cannot be certain that an audit trail really exists unless that trail is made public or is regularly reviewed by a competent and trusted authority. However, a system's operator can

always review an audit trail—indeed, this is generally the purpose of the audit trail. Our definition of accountability requires that the existence of auditing information be demonstrable to an outsider (see Section 2.1.9). Section 2.1.8 gives a more extensive taxonomy of the use of all of the technologies in this chapter for accountability.

2.1.7 Measurement of Automated Decision Systems

There has been much interest in designing tools to keep complex automated decision systems accountable by measuring and reporting their outputs, especially given differential inputs which represent sensitive distinctions between users. This work can be thought of as a kind of automatic *reverse engineering* of computerized decision making systems with the explicit goal of discovering problems with accountability.

One branch of such work develops "information flow experiments" to derive "privacy through accountability". The idea is that, by using privacy controls differentially and exposing a system to different profiles of data before observing its outputs, one can determine how those data are being used internally and whether that use comports with the stated purpose and practices of the system. The methodology for this work is described by Tschantz et al. [347], while Datta, Tschantz, an Datta report on measurement results of the system as applied to online advertising networks [112]. Datta et al. give a formal approach to assigning blame when computer systems misbehave in a way that violates a desirable property [114].

Another branch of this work considers the concrete manipulability of ranking systems such as search engines and advertising placement systems through "pollution attacks", or the introduction of targeted, inaccurate data to personalized systems to change the way they present information to an end user. Xing et al. developed a measurement system called "Bobble" for personalized search results [366] and used their observations to develop a functional method for changing search result ordering based on end-user personalization models [367]. Meng et al. demonstrate a similar

methodology for personalized advertisements [261]. Later work in the same area focused on modification of advertisements through the practice of *ad injection*, which is either incorrectly adding advertisements to web pages where there are none, or replacing existing advertisements with ones that benefit the attacker [343, 368].

Englehardt et al. attempt to discover accountability problems in websites more generally by comparing practices across sites with large-scale measurement [135]. This method has proved valuable, demonstrating the privacy implications of common practices and exposing possibly undesirable behaviors by networks of websites, as reported by Acar et al. [3] and Englehardt et al. [136].

2.1.8 Accountability

The term "accountability" has many different meanings in the computer science literature; it is used by different researchers to mean different things, and is not always defined explicitly. We provide an overview of the sorts of uses this term sees in existing scholarly work within computer science. Section 2.2.7 describes the meaning of this term in other literatures. Our own use of the term "accountability" is defined in Section 2.1.9.

A survey of approaches to accountability and uses of the term is also given by Feigenbaum et al. [146], which we follow here and supplement as necessary. Their taxonomy breaks accountability into three dimensions, "time", or when a mechanism acts; "information", or the extent to which an accountability mechanism implicates privacy; and "action", or the way in which an accountability mechanism operates. Another broad perspective comes from the perspectives of the members of a panel discussing accountability at the 1999 conference on Computer-Human Interaction, organized by Friedman and Thomas [154]. The panel responses are especially valuable to address the question of how technology can be used to improve trust in technology, a core question of this dissertation addressed further in Chapter 7.

As discussed in Chapter 1 and above in this chapter, most tools for assurance in computer science are *preventative* in nature; that is, they aim to stop misbehavior before it happens. Accountability mechanisms naturally operate in a complementary manner, enabling the review of system actions after they have taken place. Indeed, one goal of accountability mechanisms is to tie system actions to consequences if those actions are inappropriate. This tying can involve a host of mechanisms, including preventative security mechanisms such as access control measures, which allow for after-the-fact attribution of system actions. The relationship between security technologies and their interaction with humans is explored in depth in arguments by Lampson [237], who argues for the complementarity of preventative, controlling mechanisms and accountability mechanisms. Lampson also argues that because trust is a local property, systems should reject or heavily sandbox messages and inputs that are not sufficiently accountable in the sense that they don't have enough information to identify their origin if they turn out to have a bad side effect [238].

Weitzner et al. [365] also compare the preventative and accounting approaches and view them as complementary. Specifically, they relate their argument to public policy, saying:

> "[...] access restriction alone is inadequate for addressing information misuse on the Internet. An exclusive reliance on access restriction leads to technology and policy strategies in which information, once revealed, is completely uncontrolled. It's like focusing all one's attention on closing the barn door and ignoring what might happen to the horses after they've escaped. The reality is that even when information is widely available, society has interests in whether or not that information is used appropriately. Information policies should reflect those interests, and information technology should support those policies."

Weitzner et al. advocate designing systems with *policy awareness*, or a machine-readable encoding of public policy which enables compliance with stated rules. Their focus on enabling systems to reflect public policy goals is similar to ours, however we advocate the use of oversight to replace the need for policies to be encoded completely.

Risks and Failures

The earliest notions of accountability in computer science was as a collection of failure modes of deployed computer systems. Such efforts were undertaken in part to counter public perception that computers are infallible, perfect arbiters of consequential processes. A very complete and extensive collection of actual failures of computer systems has been maintained since 1985 by Neumann in the RISKS forum and its weekly digest [308]. One widely cited example of the review of failures and associated risks in automated decision systems comes from a set of accidents in the operation of the computer-controlled Therac-25 radiation therapy system. The Therac-25 gave very large radiation overdoses to at least six patients, resulting in three deaths directly attributed to the overdose. Leveson and Turner give the classic overview of the accidents, the engineering factors that led to them, and their aftermath in terms of improving safety culture and requirements in medical device engineering [245]. Another widely reviewed software failure was the explosion of the maiden flight of the Ariane 5 rocket. Lions reported on the failure for the inquiry board of the European Space Agency [248]. Many smaller failures, even with large consequences, go unnoticed and excused, however.

Distributed Systems

Accountability is used in the distributed systems literature to mean various disparate things, from the simple idea that the resource usage of a system should be measurable for billing purposes, as proposed by Reed et al. [307]; to stronger and more

complex notions that a system should "[maintain] a tamper-evident record that provides non-repudiable evidence of all nodes' actions" as proposed by Haeberlen in PeerReview [194]. Haeberlen has developed several systems under this definition of accountability, including accountable virtual machines [193], a cryptographic protocol and system for accountable randomized processes citecsar, and a proposal for accountable cloud services [192].

Other authors have considered accountability for distributed systems as the ability to associate an action with a responsible entity. Andersen et al. [10] introduce the Accountable Internet Protocol as a replacement for the standard Internet Protocol to solve this problem. AIP uses cryptography to provide *self-certifying addresses* for end Internet hosts, which allow robust information on packet origins to be learned from packet headers. A similar approach is taken, with similar goals, independently by Argyraki et al. [13].

Yumerefendi and Chase describe a framework for systems that meet accountability properties beyond the usual properties in security of *confidentiality*, *integrity*, and *availability* [370]. Specifically, they propose that accountable systems must make actions *undeniable* in the sense that they are always strongly associated with a particular actor, *certifiable* in the sense that an outside observer (a client, peer, or external auditor) can verify that the service is behaving correctly, and *tamper-evident* in the sense that attempts to corrupt the system will be detectable with high probability. Yumerefendi and Chase build a system satisfying these properties for network-available storage [372]. They also discuss the role of accountability in designing distributed systems [371].

Accountability in distributed systems is similar in flavor to research on *provenance-aware systems*, which track ownership history and a full derivation for objects they manage, a concept that originates in research on databases [74], but has grown to make a small sub-field of systems research. Muniswamy-Reddy, Holland, and Seltzer

present such a system for file storage [270] and Muniswamy-Reddy, Macko and Selzter give a related system for cloud computing [271]. A summary of similar systems can be found in Moreau et al. [268].

Cryptography

Many cryptographic systems aim to have strong accountability properties. These are particularly important for systems that make use of *anonymity* or where an actor may be known only as a member of a particular group. We describe three basic types for such systems: accountable electronic cash systems, anonymous blacklisting systems, and accountable group signatures.

Electronic cash ("e-cash") systems are perhaps the best example of the usefulness of accountability in cryptographic protocols. Such systems, introduced by Chaum in 1982 [83, 84], use *anonymous credentials* as capabilities that enable the transfer of value among users. Because many early designs required a central *bank* player to maintain information on which credentials had been redeemed ("spent"), a major design issue was *double spending*, or the multiple redemption of valid credentials. This led to a desire for systems where such malfeasance could be punished by revealing the identities of users who attempt a double spend. A later scheme by Chaum, Fiat, and Naor achieves this [87]. Later constructions by Camenisch and Lysyanskaya [76,77] go further and allow punishment by way of revoking outstanding credentials ("tokens" of value) issued to violators. Camenisch, Hohenberger, and Lysyanskaya presented another system focused specifically on accountability in e-cash systems, especially as it relates to privacy [75].

Later electronic cash systems such as Bitcoin [66, 275] attack the problem in a very different way, replacing the centralized authority that checks for double spending with a public ledger. Because all transaction flows are public, accountability is straightforward. However, in Bitcoin, transactions are designed to be irreversible,

which makes punishing bad actors a challenge. Building useful accountability systems for Bitcoin-like systems, especially while maintaining privacy, remains an important open problem.

The PEREA system of Tsang et al. exemplifies *anonymous blacklisting* systems, in which participants who commit a violation are blacklisted without being identified [345]. This is rather similar to the e-cash systems already described, although PEREA participants who misbehave are not identified—they are simply blacklisted, meaning their credentials for participation are revoked. Henry and Goldberg give a formal model of requirements for such systems [205].

Chaum and Van Heyst introduced the important concept of *group signatures* in 1991 [88], which allow any member of a group to sign a statement. Related to these are *threshold signatures* [335] and *multisignatures* [58], which require multiple participants to create. However, structures where many people sign or may have signed a message create new accountability issues that do not arise when a signature can only originate with a single signer. To address this, Micali, Ohta, and Reyzin introduced "accountable subgroup multisignatures", which explicitly have accountability as a goal and provide identifiability of individual signers from signed messages alone [264].

Many authors have considered accountability for the publication of certificates in a public key infrastructure, arguing that a PKI could *equivocate*, or present different certificates to different users (i.e., present different keys for the same name, enabling man-in-the-middle attacks). Laurie, Langley, and Kasper present a standard for cryptographically binding a certificate authority to a particular set of certificates efficiently [239]. Ryan gives a slightly more complex system with the same goals but more features [322]. Melara et al. build a similar system that aims to support large numbers of keys and large amounts of key churn, as might be expected for an e-mail or online chat service [260].

Reputation Systems

Accountability is often considered to be about assigning ownership to actions in a system, so that blame can be assigned to undesirable actions. In this way, systems for accountability are similar to *reputation systems*, which have been used to manage everything from how sensor networks should route data [73, 375] to interactions between customers of large online services [141, 310]. Extensive surveys may be found in Resnick et al. [309] and Jøsang, Ismail, and Boyd [217].

Accountability and Privacy

Accountability is often used in the context of discussing privacy, since many privacy problems arise from problematic uses of data, which cannot be directly and verifiably controlled in the same way that collection can. Verifiability is the more important problem—a company with lots of private customer data may well follow very good practices, limiting disclosure of data to only those people and services within the company that actually need them, storing the data encrypted when they are at rest, and otherwise following accepted best practices for handling data. However, these practices cannot be verified by an end user or outsider without significant extra effort on behalf of the data holder. Pearson and Charlesworth argue that procedural and legal solutions, as a complement to appropriate technical measures, can provide the necessary accountability to convince end users that their data are being handled in appropriate ways. Formal models for privacy protection and accountability are summarized below. Datta provides a survey of this literature from a computer science perspective [113].

From a more theoretical perspective, Feigenbaum et al. [146] examine the notion of accountability as responsibility and blame for its privacy implications, asking whether, in bringing accountability to a system might damage privacy if it requires more robust notions of identity or reveals information about the behavior of actors in the system,

either in the case of a violation or even during normal operation, especially if such information is disclosed beyond the system, either publicly or to an authority.

Formal Models

Many authors have described formal models of accountability and responsibility. Feigenbaum, Jaggard, and Wright give a formal model similar to the one we use, where system actions are recorded in *event traces* and these event traces are reviewed for correctness automatically or by an authority [145]. Feigenbaum et al. postulate that system operators respond to *utility functions* that trade off their mission-oriented objectives with the cost of punishment for inappropriate action. Feigenbaum independently analyzed the question of whether such a formal model can be usefully deployed to address real world political oversight issues [144].

Barth et al. [29] give a formal model relating privacy practices in business processes to notions of utility for data use, offset by punishment for inappropriate uses. Separately, Barth et al. define a formal model for *privacy as contextual integrity*, a philosophy of privacy which views information use as being morally tied to the context of the information's collection and use [28]. Tschantz, Datta, and Wing give a formal model for identifying restrictions on information in privacy policies, useful for defining compliance predicates for system oversight [348].

Küsters, Truderung, and Vogt give a formal model for accountability in cryptographic protocols, relating it to *verifiability*, or the property that some invariant can be checked. Their model, which focuses specifically on a process's goals, separates the idea that an invariant is verifiable from accountability for that invariant, a stronger notion that requires the ability to make a *judgment* that blames a specific party if the invariant is violated [235]. Datta et al. give a more detailed formal model of how to assign detailed blame to specific program actions [114].

Jagadeesan et al. give a formal model of accountability through after-the-fact auditing based on blaming individual participants in a distributed system for violating some desirable invariant [212]. This is similar to the Barth et al model of privacy auditing for business practice compliance [29].

Bella and Paulson give a formal model of accountability protocols in which protocols must produce for each participant "evidence, typically digitally signed, about actions performed by his peer" [33]. They give implementations of two concrete protocols along with a formalization of those protocols in their model which is proved correct in the Isabelle theorem prover. The ultimate accountability goal is to bind actions to identities. Similarly, Kailar gives a logic for analyzing accountability properties of protocols and considers example applications of this logic to some electronic commerce protocols [219]. Kailar's framework focuses on accountability as the ability for a third party to associate the originator of an an action to the action. Backes et al. also used protocol logics similar to classical authentication logics to investigate accountability properties in contract-signing protocols [20].

Tamper-Evident Systems

Tamper evidence is another approach taken by computer scientists to providing accountability. It is useful especially in producing audit trails which can be verifiably shown not to have been modified. A general technique for this was suggested by Crosby and Wallach [109]. Suh et al. describe AEGIS, an architecture for tamper-evident computing on systems where only the processor is trusted [342]. Waldman, Rubin, and Cranor describe Publius, a system that uses tamper-evidence properties to guarantee censorship resistance [358].

Engineering Ethics

In an effort to provide accountability for professional members of the technology industry, the Association for Computing Machinery (ACM) and the IEEE Computer Society jointly adopted a code of professional ethics, after many years of discussion in the computer science community [178, 179].[7] Leventhal, Instone, and Chilson report on their study of attitudes towards ethics among computer scientists and relate these attitudes to a comprehensive review of literature on ethical practices in the field [244]. Johnson writes more broadly about ethical issues involving computers and computerization [215]. Narayanan and Vallor argue that, if we want to expect professionals in the technology industry to be equipped to act ethically, ethics must become a formal part of computer science curricula at the university level and before, despite the fact that this has not yet happened [285].

2.1.9 Our Definition of Accountability

The Oxford English Dictionary defines "accountable" as

> "Chiefly of persons (in later use also organizations, etc.): liable to be called to account or to answer for responsibilities and conduct; required or expected to justify one's actions, decisions, etc.; answerable, responsible."[8]

In particular, the dictionary gives "answerable" as a close synonym. This is consistent with the definition given by the Merriam Webster dictionary for "accountable" as "required to explain actions or decisions to someone" or "required to be responsible for something". Both definitions focus on the idea of accountability as the ability to explain actions to an oversight body or to the public, leaving implicit the question of whether accountability implies consistency with the legal, political, or social norms of

[7] The most recent version of the code can be found at `http://www.acm.org/about/se-code/`.

[8] "accountable, adj." OED Online. Oxford University Press, June 2015. `http://www.oed.com/view/Entry/1198`

those to whom a process or entity is accountable. For the purposes of this dissertation, we use a slightly stricter definition of accountability which requires this consistency.

Definition 2 (Accountable)**.** *We say that a process or entity is* accountable *for a decision if that process is consistent with the* legal, political, and social norms *that define its context, and this fact is publicly evident, either because such consistency can be readily determined using public information or because an entity empowered and trusted to define such consistency (e.g., a court, a legislature, or other designated authority) can be shown to have the necessary information to determine this consistency.*

In particular, we consider in this dissertation the setting of a *decision authority* making decisions using a set *policy* which are particularized to a set of *decision subjects*. In this setting, under our definition of an accountable decision process, all of the following properties are *necessary*:

(i.) the authority must be committed to its policy in advance;

(ii.) the result asserted by the authority must be the correct output of the authority's committed policy when applied to an individual decision subject;

(iii.) any randomness used in the decision policy must be generated fairly; and

(iv.) if necessary, the authority can reveal its policy to an oversight body for examination later, without revealing decision subject's private data, and that body as well as each decision subject can verify that the revealed policy is the one that was used to make the decisions in question.

The question of what properties are *sufficient* to define an accountable process is difficult and, being an inherently political question, possibly un-answerable. We attempt an answer in Chapter 7, where we describe how to design automated processes so that they are accountable under the definition given here.

2.2 Related Concepts from Fields Outside Computer Science

In this section, we summarize concerns related to the governance and oversight of computer systems and algorithmic decision processes as they have been conceptualized in literatures beyond computer science. This discussion has largely been centered around the concept of *transparency*, the idea that the design of and inputs to the computer system (or at least assertions about them) can or should be made either to an oversight body or to the public. A special notion of transparency that has gained particular purchase for its similarity to established social science techniques is *reverse engineering*, described in Section 2.2.4. Automated methods for measuring and thereby reverse engineering the decisions of complex decision systems have been described by computer scientists, as we describe in Section 2.1.7. Beyond transparency, others have called for aggressive *oversight* or *enforcement* of existing legal requirements, as we describe in Section 2.2.5. Current doctrine generally protects two categories of legal harm that might be inflicted by inadequately governed computer systems: violations of the *due process* rights of decision subjects (described in Section 2.2.2) and problems with *fairness*, either of the decision outcomes or the process itself (described in Section 2.2.6).[9] We start in Section 2.2.1 with a broader framing of governance problems, including ancient and modern scholarship on the meaning and function of rules in a society, and general approaches for addressing them. We finish this section with a discussion of the use of the term "accountability" as it applies to automated systems when described outside of computer science. For each concept in this section, we begin by explaining the concept and its history broadly and then turn to related literature specific to governing automated decision systems.

[9]While scholars of due process have argued that due process rights can also protect against certain substantive notions of unfairness, we separate out, somewhat arbitrarily, the notion of procedural regularity, which the tools introduced in Chapter 5 are especially suited to assuring.

2.2.1 Philosophy of Law, Rule of Law, and Software as Law

Philosophers of law and government have long postulated that consequential processes should be governed and should be accountable to a well defined set of rules and laws. The general principle is that well defined rules create legitimacy, since those affected by some process do not feel that their treatment was arbitrary. The main lesson from this scholarship is that the rules under which a process operates should be well understood and substantially complete in advance but also flexible enough to accommodate any unexpected situation. This gap is bridged by a *judge* who is politically empowered to determine how rules apply in a given case but who is also politically constrained to uphold the rules. For example, Aristotle wrote in *Politics* that "it is more proper that law should govern than any one of the citizens", arguing that *rule of law* is a supreme principle of effective governance. Aristotle also recognized that complete rules cannot be laid out in advance and that some discretion and oversight by a judge or other accountable person is necessary, saying "the law having laid down the best rules possible, leaves the adjustment and application of particulars to the discretion of the magistrate" [14]. Later, these ideas were coalesced by (among others) Hobbes in *Leviathan* [208], Locke in *Second Treatise of Government* [252], and Rousseau in *The Social Contract* [320] as *social contract theory*, in which governance processes achieve trust and legitimacy through "the consent of the governed". That is, people come to trust consequential processes because they can hold those processes accountable by revoking their consent to participate.

These ideas wound their way through the philosophers and founding documents of Western civilization (including the 1215 Magna Carta and the 1787 United States Constitution)—a summary of this development is beyond the scope of this section—to reach modern-day legal scholarship and jurisprudence. There are several contemporary schools of thought on the purpose and function of law. We summarize a few relevant theories here; a full exposition of these theories is beyond the scope of this

dissertation. A summary of the meaning of rule of law across modern legal systems can be found in Bingham [47].

One relevant framing of how law should govern consequential processes is given by Lon Fuller in his widely cited *The Morality of Law* [156]. Fuller describes the goals and purposes of law and legal systems in terms of eight possible modes of failure:

(i.) The lack of rules or law, leading to *ad hoc* decisions and inconsistent adjudication.

(ii.) Failure to publicize or make known the rules of law.

(iii.) Unclear or obscure legislation that is impossible to understand

(iv.) Retrospective legislation, i.e., legislation that applies new rules to past facts and circumstances.

(v.) Contradictions in the law.

(vi.) Demands that are beyond the power of the subjects of the law to fulfill.

(vii.) Unstable legislation, i.e., legislation that changes too often to be meaningfully disclosed or understood.

(viii.) Divergence between adjudication or administration and legislation.

Accountable algorithms address many of these concerns directly, and can address all of them in concert with other technical tools from both computer science and the theory and practice of governance and oversight. In this way, the techniques described in this dissertation can be deployed to reconcile the perceived gap between the governance of computerized decision making systems and systems operated by humans. See Chapter 7 for a more detailed explanation of this argument.

Fuller sees the moral effects of law as the source of law's binding power, imbuing the social contract with a moral dimension. This stands in contrast to alternative *positivist* theories of the law, such as H.L.A. Hart's famous counterpoint to Fuller [203].

Positivists view laws as a social construction, arguing that laws exist independently of their merit or whether any system of law conforms to a "natural" or "fundamental" notion of justice, democracy, or rule of law. Early positivists framed of rules as commands which exist independently of any fundamental normative grounding. This maps well onto many critiques of automated decision systems: critics of automated decision systems often argue that such systems are consequential and have the power to affect or control the interaction between people and the rest of the world, but that automated systems exist independent of normative or legal frameworks.

Yet another school, *interpretivism*, argues that the legally significant actions and practices of political institutions can modify the rights and obligations of others under the law. Interpretivism is famously associated with Ronald Dworkin [132, 133].

Finally, scholars of legal realism argue that law is operationalized through fundamentally political institutions and therefore is not independent of political discourse.

Developing a framework relating the function of law to automated processes, Lessig, in his seminal *Code* and its follow-up *Code: Version 2.0* [242, 243], develops the idea that software itself can regulate personal conduct and that such regulation is complementary to legal regulation. Lessig argues that the two kinds of regulation are not incompatible, but rather part of a larger scheme of regulatory forces that include law, norms, software code, and markets. Lessig suggests that, while many activities are regulated by software, this regulation is ultimately subordinate to a larger regulatory framework that includes laws, norms, and markets as well. Lessig refers to software code as a kind of law and suggests that choices in the design of computer systems can create "Architectures of Control", coded into products, which shape and become part of the way people interact with the world.

2.2.2 Due Process

Another related concept from the law is that of due process, the idea that rights, as cognized by the legal system, cannot be stripped or ignored except under specific, preconceived legal processes. Like the philosophy of law, due process has a long history in legal scholarship: it dates back at least to the 1215 Magna Carta, which stipulates that "No free man shall be seized or imprisoned, or stripped of his rights or possessions, or outlawed or exiled, or deprived of his standing in any other way, nor will we proceed with force against him, or send others to do so, except by the lawful judgment of his equals or by the law of the land."[10] The United States constitution provides similarly in the Fifth and Fourteenth amendments: "[N]or shall any person ... be deprived of life, liberty, or property, without due process of law ..."[11] Courts have interpreted this phrase to protect *procedural regularity* in civil and criminal legal process, or the idea that all people subject to some process must be subject to the same version of the process and be treated according to the rules of the process. In certain cases, due process is also interpreted to protect well as certain substantive forms of fairness, but our focus will be on procedural regularity. The tools introduced in this dissertation (specifically, in Chapter 5) are very well suited to assuring procedural regularity. Additionally, it is fundamentally impossible to provide assurances of a substantive invariant to subjects of a decision process unless those subjects already have assurance that the process satisfies procedural regularity. We address the enormous topic of substantive notions of fairness in Section 2.2.6. Section 2.1.4 addressed how fairness has been conceptualized within computer science.

Some of the earliest work on oversight of automated decision making comes from Danielle Keats Citron's article "Technological Due Process" [94], which investigates

[10] Text of the Magna Carta, `http://legacy.fordham.edu/halsall/source/magnacarta.asp`.

[11] Text of the United States Constitution, Amendment V. Amendment XIV reads similarly, but applies the requirement to the states individually. The full text is available online through the Cornell Legal Information Institute, `https://www.law.cornell.edu/constitution`.

whether and how automated processes satisfy the requirements of due process. Citron argues that the traditional, pre-computerization approaches to making and adjudicating rules are "rapidly becoming outmoded" because traditional procedural safeguards are circumvented both by the way computer systems are developed and in the way they are applied in particular cases. This core argument stems from a reflection on case studies of failures in automated systems for determining public benefits. These failures, in turn, resulted from a combination of inadequate specification of the computer programs in question, inadequate testing, and inadequate oversight to discover and ameliorate the problems before the systems were deployed for real use. Citron argues that such problems can be violative of due process rights, since the rules ultimately used by the computer system differed in substance from the rules duly enacted by the policy process, meaning that they were not subject to the required public review and comment processes. Citron extends some of these same critiques in later work with Pasquale [97] that examines private-sector decision making. However, due process as a legal concept applies primarily to actions taken by the state. Still, although it may not be legally required, a certain modicum of uniformity in process and procedural regularity is desirable even for privately ordered decisions, as we will investigate in Chapters 5 and 6.

Pasquale also addresses due process in automated decision making systems as part of a larger investigation of the nature of governance for such systems. He regularly mentions due process as a goal of such governance, and suggests that due process rights and obligations could be applied to private operators of computer systems as well as to the government, a regime which is not without precedent [294].

2.2.3 Transparency

A straightforward solution to the problem of verifying procedural regularity is to demand transparency: if the code and input-output pairs are public, it seems easy to

determine whether procedural regularity is satisfied. Indeed, transparency or partial transparency of computer systems can be a helpful tool for governance in many cases, for example for televised lotteries, which have completely public rules and which pick verifiably random inputs using a publicly viewed ceremony. However, as we will explain in this section, relying on transparency alone is a naïve approach that cannot provide accountability in all situations.

Transparency has often been suggested as a remedy to accountability issues for automated decision making systems. In his book *The Black Box Society* [294], Pasquale calls directly for operators of automated decision systems to provide detailed information on their behavior so as to provide checks on socially unacceptable outcomes. Citron also suggested that transparency could alleviate the due process problems she identified in government benefits systems [94] and systems that rate consumers for credit or other business purposes [97]. Citron also argues that all software that makes consequential administrative decisions must be open sourced [95].

Despite these many calls for transparency as a sovereign remedy to problems of accountability in automated decision making systems, however, transparency alone is not sufficient to provide accountability in all cases. Transparency is useful and often extremely valuable, but does not serve the goals of its proponents entirely or without other techniques. The tools introduced in Chapter 5 rely on partial transparency (that is, transparency of only certain facts, or verifiable transparency only to certain designated parties) to achieve accountability. Chapter 7 describes in more detail how these tools, and partial transparency in general, provide for accountable automated processes.

First and foremost, it is often necessary to keep the source code for computer systems and their data secret to protect business interests, privacy, or the integrity of law enforcement or investigative methods. Secrecy discourages strategic behavior by

participants in the system and prevents violations of legal restrictions on disclosure of data.

Second, while transparency allows for static and at least some dynamic analyses of source code (described in more detail in Section 2.1.2), those methods are often insufficient to verify properties of programs if those programs have not been designed with future evaluation and accountability in mind.[12]

Third, transparency does not guarantee that an output can be replicated. Computer systems often interact with their environment, for example by examining other running processes, reading the time of day, accessing a changing data store, or incorporating user input. Further, many algorithms require randomness, and are usually implemented by polling a system device (e.g., a piece of hardware, a software subsystem designed to provide high-entropy bits such as `/dev/random` or `/dev/urandom` on UNIX-like systems). If the environmental factors encountered by a computer system cannot be precisely replicated, then the output cannot be replicated either. An easy solution is to redesign an automated process to make any environmental interaction an explicit input, which works as long as the input can be recorded when the system is run. However, this does not address the problem of determining that the input recorded is a correct reflection of the environment or satisfies some other property, such as being a randomly chosen bit string. In Chapter 3, we survey various approaches to the problem of providing verifiable random bits from the computer science literature. Still, verifiability can easily suffer unless care is taken. With a naïve approach, a corrupt decision maker could construct a plausible but mis-representative audit trail: for example, a system implementing a random choice could be run multiple times until the desired outcome was reached, and then only the record of the

[12]In particular, Rice's Theorem [311] states that it is undecidable to determine any nontrivial property of a program, for a very general definition of "nontrivial". More precisely, given a set of languages S, S is nontrivial if there exists a Turing machine that recognizes languages in S and there exists a Turing machine that recognizes languages not in S. Then the theorem states that it is undecidable to determine if the language recognized by an arbitrary Turing machine lies in S.

final run kept for audit purposes. A simple lottery provides an excellent concrete example of this problem: a perfectly transparent algorithm—use a random number generator to assign a number to each participant and have the participants with the lowest numbers win—yields results that cannot be reproduced or verified because the random number generator will produce new random numbers when it is called upon later.

Fourth and finally, systems that change over time cannot be fully understood through transparency alone. System designers regularly change complicated automated decision processes—search engine rankings, spam filters, intrusion detection systems, or the algorithms that select website ads—in response to strategic behavior by participants in the system. The computer systems that choose which social media posts to display to users might respond to user behavior. "Online" machine learning systems can update their internal predictive model after each decision, incorporating new observations 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.

Pasquale argues a realist version of this last point, observing that transparency requirements are often defeated by making the processes themselves more complicated so that disclosures cannot be properly understood. He argues that this has happened in response to disclosure regimes in the financial industry and for information privacy practices online.

Accountable algorithms make use of transparency to achieve accountability, but in a focused way, making transparent the things which may be published while using cryptography to verify the correctness of the things which cannot be. Further, accountable algorithms use cryptography to limit the disclosure of facts about a systems to parties that are authorized to learn them, while providing guarantees to the public that this disclosure has been performed correctly and that the correct facts have been

disclosed (e.g., that the decision policy disclosed to an oversight body is the policy used to make decisions about particular users). We give more detail on this argument in Chapter 5.

2.2.4 Reverse Engineering

For many authors, the most obvious approach to holding automated decision systems to account is to reverse engineer their behavior, so as to test them for unacceptable behavior. This approach is similar in its goals to the approach of transparency, described in Section 2.2.3, in that it attempts to examine the acceptability of a system's behavior based on its operation, specifically based on its input-output relation. Reverse engineering can be viewed as a form of forced transparency or a way of imposing transparency on systems that do not disclose the nature of their operation in detail.

Reverse engineering is espoused as a solution to problems of oversight and accountability by many authors in many fields. We describe the use of measurement and reverse engineering techniques for complex automated decision making systems as applied within computer science in Section 2.1.7. Here, we summarize the arguments made by proponents of reverse engineering automated decision systems by scholars in other fields.

Diakopoulos [121] describes the reverse engineering of several specific computer systems and considers the applicability of this methodology for journalists more generally, observing that it is difficult to sufficiently test systems due to lack of human and computational resources. Further, testing only specific inputs and outputs gives a limited view of the correctness or rightness of the operation of a particular computer system and its deployment in the real world. Diakopoulos sees computer systems as inherently related to their human creators, and believes that any attempt to consider the accountability of such systems must consider the intent of anyone involved in the system's design or deployment as well as the agency of anyone who interacts with

the system in a decision making capacity, even those who merely read off and act on a system's output. Diakopoulos recognizes that transparency is an imperfect solution to holding automated decision systems accountable, noting several situations where the interests of the operator of a system conflict with the interests of those who would benefit from transparency, such as when a system has legitimate secrecy goals or when a decision maker's interests, narrowly construed, oppose the interests of decision subjects. However, Diakopoulos embraces transparency when it is possible and reverse engineering when it is not as enabling tools for journalists to review important systems deployed in practice.[13]

Pasquale [294] argues in favor of using reverse engineering to examine the actual behaviors of complicated systems such as search engines, advertising exchanges, credit scoring systems, and systems which automatically trade financial securities. In other work with Citron [96], the authors advocate reverse engineering, or at least observation, for tracking the flow of sensitive intelligence information between collaborating agencies and through government and private-sector collaborations as a method to create accountability.

Sandvig [327] argues that unacceptable behavior by automated decision systems could be discovered by use of the auditing methodology from social science, in which researchers who cannot control a process carefully control the input data and observe differential results, effectively forcing the transparency of a system's input/output relation. Sandvig applies this methodology to the Facebook news feed, in an attempt to determine how posts are selected for viewing. This sort of auditing for reverse-engineering unacceptable behavior has also been automated on a larger scale by computer scientists to investigate the manipulability of search engine rankings and the sorts of targeting performed by advertising systems, as described in Section 2.1.7.

[13] A more detailed version of Diakopoulos' argument is given in his report for the Tow Center for Journalism, `http://www.nickdiakopoulos.com/wp-content/uploads/2011/07/Algorithmic-Accountability-Reporting_final.pdf`.

However, as noted in Section 2.2.3, we do not believe that transparency alone—whether voluntary on the part of a decision maker or introduced forcibly by the use of reverse engineering—is sufficient to provide accountability. Indeed, this method of "auditing algorithms" can be viewed as a form of black-box dynamic end-to-end testing, a very weak form of evaluation. Instead, we advocate achieving accountability by designing important decision processes for oversight by a politically empowered authority, as described in Chapter 7. The techniques of Chapters 4 and 5 are valuable tools for such designs, and can enable evaluation that is similar to white- or grey-box testing. Reverse engineering, or at a minimum the monitoring of actual outputs of such systems, can be a useful tool for designing oversight-driven processes, however, especially in cases where the process under investigation is hostile to oversight or where no trusted oversight body exists.

2.2.5 Oversight and Enforcement

As mentioned, the concept that one governing body should have jurisdiction to review another is fundamental to the founding documents of modern Western civilization. The famous "security clause" of the Magna Carta, signed in 1215 to check the power of the king, allowed a council of 25 barons to notify the king when an official action transgressed the limits of acceptable behavior and for the barons to seize the king's assets (by force, if necessary) if he did not make amends to their satisfaction within a fixed period of time. The authors of *The Federalist Papers* drew on this history to argue in favor of the ratification of the United States constitution, supporting strongly the notion that different functions of government should be separated into different institutions, with these institutions holding formal checks and balances against each other [319, Papers 47–51]. Judicial review of statute law followed from these documents in the United States [258], but not in the United Kingdom. Meanwhile, the United States Congress exercises regular oversight hearings and reviews government

operations through its system of committees. The constitution only implies powers and duties of congressional oversight and investigation, however—these powers are not enumerated functions of Congress.[14]

On the executive side, many agencies are tasked with enforcing existing laws against private citizens and corporations. For a model of how executive agency decisions relates to the congressional oversight process, see Weingast and Moran [364].

For automated decision making, the Federal Trade Commission is perhaps the most salient, although other agencies, such as the Securities and Exchange Commission or the Consumer Financial Protection Bureau may have jurisdiction depending on the nature of the decision. Notably, the FTC has expressed interest in reviewing the fairness of automated decision making, having held a workshop on the impact of big data decision making processes on traditionally marginalized communities.[15]

Expressing concerns similar to Citron's about the due process implications of automated decision making, Bamberger argues that the emergence of automated systems to judge a company's compliance with the law can lead to a gap between what a court would find (non)-compliant and the judgment made by an automated system when programmers interpret the law while designing and developing computer systems, [24]. Bamberger argues that law enforcement (or at least efforts for legal compliance within organizations) by automated decision processes is inherently lacking, as the technologies that enable it are not, and cannot be, vested with the same level of trust as political institutions.

Citron [94,97] and Pasquale [293,294] argue similarly for the inadequacy of current enforcement regimes. Pasquale in particular calls for stronger penalties and normative regulations for automated decisions, in contrast to the current model under which

[14]For an overview of the congressional committee system, its powers, and their authorities, see the Final Report of the Joint Committee on the Organization of Congress from December, 1993: `http://archives.democrats.rules.house.gov/Archives/jcoc2.htm`.

[15]See an agenda and complete video of the event at the FTC's website: `https://www.ftc.gov/news-events/events-calendar/2014/09/big-data-tool-inclusion-or-exclusion`.

decision makers simply assert their policies and are typically punished only for violating their self-described promises. Pasquale is particularly critical of industry "self regulatory" regimes, under which many players in the same industry cooperate to produce a set of "best practices", or standardized promises [295].

Accountable algorithms improve and complement current regimes for oversight and enforcement by enabling oversight bodies and law enforcement agencies to precisely relate the facts of a particular decision to a *justification* for that decision held in an audit trail (i.e., the decision policy in force for that decision and the inputs it was given). Further, accountable algorithms allow this while reducing the amount of private data that must be reviewed by the overseer or law enforcement agency. Finally, accountable algorithms can demonstrate that the facts disclosed to an overseer or during a law enforcement investigation correspond to the decisions announced to individuals, investing the law enforcement and oversight process with additional legitimacy.

2.2.6 Fairness

Fairness is a deep and subtle subject addressed by an enormous literature in philosophy and the law, covering notions of justice, equity, social organization, the distribution of goods and services in a society or organization, the norms of interpersonal interaction, and the relationship between humans and the environment. Any review of this vast literature would necessarily be incomplete, so we content ourselves to mention only a single, relatively modern framework: the *Theory of Justice* by Rawls [306]. We do not rely on Rawls specifically, but the theory provides a useful framework for discussing questions of fairness for the purpose of this section.

Rawls' theory is often known as "Justice as Fairness" and deals with the problem of distributive justice, or the allocation of goods in a society. Rawls draws on a variant of the social contract, described above, defining his theory via two principles of justice:

(i.) the *liberty principle*, which states that "First: each person is to have an equal right to the most extensive basic liberty compatible with a similar liberty for others"[16]; and (ii.) the *difference principle*, which states that social and economic inequalities should be arranged so that they are to be of the greatest benefit to the least-advantaged members of society and that offices and positions must be open to everyone under conditions of fair equality of opportunity. Rawls develops these principles through an artificial hypothetical device he calls the *original position*, in which the members of a society decide together on the principles of justice their society will use from behind a *veil of ignorance*, without knowing their position in society, their class or social status, or their distribution of assets or natural abilities. Rawls argues that, from behind the veil of ignorance, every person should choose principles of justice that are fair to all, specifically, principles consistent with the liberty and difference principles. Rawls notes that "[these principles] are the principles that rational and free persons concerned to further their own interests would accept in an initial position of equality as defining the fundamentals of the terms of their association." Critiques of this theory have focused on its inability to explain observed and embedded inequalities in modern societies, its lack of accounting for personal injustices, and its use of game-theoretic models without sufficiently justifying parameter choices, thereby weakening the strength of its conclusions.

Automated decision making systems are superficially Rawlsian in that they are designed and developed prior to entering the real world. However, such a parallel analysis does little to explain systems that react to their deployment environment through their designers and maintainers, as nearly all systems do. Also, Rawls' original position argument is about the design of entire societies, and does not apply

[16]The liberty principle is superficially similar to Kant's categorical imperative, set forth in his *Groundwork for the Metaphysics of Morals*. Rawls' synthesis of this principle with notions of equality and his elevation of moral reasons as tools that can override non-moral reasons are also Kantian. Rawls differs from Kant and his other antecedents in that he conceives of liberties as being constructed socially, rather than by individual choices.

directly to the design of specific systems which might advantage or disadvantage particular people. However, the original position provides a strong moral grounding that such systems should be designed to minimize inequalities and general harm to those least well off in society. We return to the question of how to design systems for effective governance and the relationship between such governance and social acceptance of such systems in the vein of the social contract in Chapter 7.

However, much has been said about the fairness or unfairness of automated decision making systems specifically, particularly about the ways in which unfairness in those systems may be hidden unintentionally, masked by a malicious actor, or may lack a legal remedy because the structures governing current automated decision making do not adequately cognize the possible harms. An exemplar of all of these criticisms of automated decision systems, specific to systems that make predictions using so-called "big data" techniques, or machine learning, comes from Barocas and Selbst [26]. Proponents of such systems often argue that because models used to make decisions are simply revealing objective patterns in accurate data, they must be neutral. Barocas and Selbst provide several counterpoints to this reasoning. For example, they argue that such systems can easily find discriminatory patterns by learning the salience of variables that are highly correlated with a protected status attribute.[17] Second, biases of the designer of the underlying model may come through in the form of various choices made in building the model, such as the selection of what modeling approach to use, what values to optimize, what training data to use, the choice of which feature set from those data to use, or even malicious choices intended to allow

[17]The once-common, now-illegal practice of *redlining*, or evaluating the creditworthiness of a loan applicant based on the address of the property pledged to the loan, is an example of such a "proxy" or holographic encoding of a protected status attribute in an otherwise unprotected feature. Redlining was overtly a method of reducing lending to racial minorities, and its impact on communities and cities can be easily seen simply by reviewing the "security" maps of the depression-era Home Owners Loan Corporation, a public-private partnership intended to guarantee loan refinancing during the Great Depression, said to have originated the practice. See `http://www.urbanoasis.org/projects/holc-fha/digital-holc-maps/` for a sample of such maps.

innocuous explanations for discriminatory decisions. Barocas and Selbst consider the legal doctrine of *disparate impact*, which supplies a remedy when a formally neutral process nonetheless discriminates by having a systematically unequal impact on a legally protected group, such as a racial minority, a particular gender, or a religious group. They argue that disparate impact does little to ameliorate these harms as it is currently constituted because automated decision making systems fail to meet the appropriate tests set forth in anti-discrimination law. They use systems that screen applicants for jobs, now commonly used for the initial review of candidates, as a running example for their legal analysis, primarily because hiring discrimination has the most mature precedential case law.[18] A similar analysis but of the impact of big data decision making on civil rights in the financial sector comes from Robinson, Yu, and Rieke [315].

In a similar vein, Citron and Pasquale offer a critique of the opacity of systems for assigning scores to consumers, which they say violate norms of due process because the consumers who are being scored cannot understand how the scores are determined and cannot effectively audit or challenge the process by which the scores are determined [97].[19] They suggest that the appropriate remedy for these problems is

[18]For an overview of such systems and their trade-offs, see a review of the approach in The New York Times by Claire Cain Miller, `http://www.nytimes.com/2015/06/26/upshot/can-an-algorithm-hire-better-than-a-human.html`. Miller also summarized algorithmic discrimination more generally in a later piece, `http://www.nytimes.com/2015/07/10/upshot/when-algorithms-discriminate.html`.

[19]A more detailed description of various consumer-focused scoring systems can be found in the World Privacy Forum's report *The Scoring of America: How Secret Consumer Scores Threaten Your Privacy and Your Future* by Pam Dixon and Robert Gellman. See `https://www.worldprivacyforum.org/category/report-the-scoring-of-america/`. Similar concerns were echoed in the White House's report on big data, *Big Data: Seizing Opportunities, Preserving Values*, written by the President's Review Group on Big Data and Privacy, led by Counselor to the President John Podesta, available online at `https://www.whitehouse.gov/sites/default/files/docs/big_data_privacy_report_may_1_2014.pdf`. In the report's opening letter to the President, the review group remarked "A significant finding of this report is that big data analytics have the potential to eclipse longstanding civil rights protections in how personal information is used in housing, credit, employment, health, education, and the marketplace." The report was accompanied by a report by the President's Council of Advisers on Science and Technology, *Big Data and Privacy: A Technological Perspective*, which was somewhat friendlier to the commercial uses of data. See `https://www.whitehouse.gov/sites/default/files/microsites/ostp/PCAST/pcast_big_data_and_privacy_-_may_2014.pdf`.

a regime of procedural safeguards, including regulatory oversight, transparency to facilitate testing by regulators, robust audit trails that facilitate notice to consumers and regulators of the system's actual actions, and the opportunity for individuals to engage interactively with the system to learn how it will treat specific facts and circumstances. A similar critique is offered by Crawford and Schultz [106], who argue for a right to "procedural data due process", a framework much like that of Citron and Pasquale in that it demands notice to consumers and oversight entities, the right to a hearing on the correctness of outcomes, and the right to impartial judicial review.

A philosophical framework for analyzing these discrimination problems is provided by Friedman and Nissenbaum [153], who give a detailed taxonomy of possible biases, framing various kinds of bias as pre-existing in society (e.g., due to the biases of individuals involved in the design or due to biases accepted by the organizations, institutions, and societies that field a particular computer system), technical in nature (e.g., due to faulty random number generation or due to the use of an algorithm designed to work well in one context being used in a different, incompatible context), or emergent from the use of a system (i.e., due to unforeseen interactions between a system and its environment). They provide a litany of examples of computer systems that exhibit biases and a relation of those problems to their taxonomy. The goal of this analysis is to provide a purely theoretical view to whether and why a system is biased, without consideration of whether or how that bias should be redressed by the law or society in which the system under consideration operates. Friedman and Nissenbaum argue that the level of bias in a computer system should be a criterion that is evaluated as part of its suitability for a particular use and one that should be considered at the design stage.

Our approach differs from this previous work: our protocol generates a robust audit trail as called for by Citron and Pasquale and by Crawford and Schultz and facilitates the kind of oversight that would enable a robust anti-discrimination liability

regime as envisioned by Barocas and Selbst. By enabling effective oversight, our approach forces system designers to consider the fairness of their systems at the design stage, so that they can justify the operation of their systems during later oversight, as desired by Friedman and Nissenbaum. We discuss how to define or certify fairness in a computer science context in Section 2.1.4, give our protocol itself in Chapter 5, and explain how it answers the questions in the above-mentioned work by providing a way to design systems to admit robust oversight in Chapter 7.

2.2.7 Accountability

Accountability is not a unified concept in philosophy or law, but rather exists in the penumbra of many topics already discussed above such as due process, oversight, the separation of powers, judicial review, and law enforcement. We discuss here uses of accountability in the law, communications, sociology, journalism, and philosophy literatures; in Section 2.1.8 we survey the uses of the term in computer science and give our working definition of accountability for this dissertation.

Nissenbaum [287] gives a broad conceptual framework for the accountability of computerized processes. She identifies four barriers in automated systems to robust accountability: (i.) the problem of many hands, or issues related to many people being involved in the production and deployment of an automated system such that no one person can be blamed for a problem; (ii.) the problem of bugs, especially given that the field has come to view them as routine and unavoidable; (iii.) blaming the computer, or the use of an automated system as a scapegoat for undesirable behavior (on the theory that the system is operating as designed and is unchangeable, so the undesired behavior is really intended); and (iv.) the ability to own and operate software without being liable for its errors, while at the same time intellectual property law gives many rights to software developers without assigning them concomitant responsibilities. Nissenbaum does not feel that these factors, even taken together, imply the automated

systems cannot be accountable. Rather, she argues that they obscure accountability, and that accountability can be achieved with the proper standard of care, as it has been in other areas suffering similar difficulty. Nissenbaum argues for a regime of *strict liability*[20] and strong responsibility for software producers.

Halpern and Pearl describe a conceptual framework that relates actions by a system to responsibility and blame [200, 201] that was built on by Chockler and Halpern to give a detailed framework for responsibility in artificial intelligence systems. While responsibility and blame do not completely define accountability, they are important building blocks.

Grant and Keohane [181] provide a thorough evaluation of accountability from a political science perspective. They examine the use of accountability as a deterrent mechanism against abuses of power in global politics, examining whether and when it is effective at the level of individuals, organizations (NGOs, multilateral bodies, corporations, trans-governmental networks), or nation-states. They identify seven types of accountability mechanism and examine when each is applicable and at what level. Mulgan gives another detailed political science analysis of accountability, focusing on public administration in democracies specifically, arguing that it is a poorly defined concept without agreed-upon definitions or boundaries [269].

Citron and Pasquale investigate the accountability of information sharing practices in so-called "fusion centers", or partnerships between law enforcement, the intelligence community, and private industry created to enable the rapid collection, integration, and interpretation of intelligence data on threats to public safety [96]. They determine that information practices often skirt the boundaries of the law when viewed in the aggregate, but that no single agency or entity can obviously be blamed for violating a particular rule. They argue that a new form of *network accountability* is necessary to govern these practices, which takes into account the actions of the entire group

[20] A legal term of art, strict liability refers to a regime where the producer of a good is liable for defects in that good even when they have taken reasonable care to avoid them.

of participants when evaluating whether a rule has been violated and apportioning blame.

Chapter 3

Required Cryptographic Primitives

In this chapter, we introduce the cryptographic primitives necessary for the protocols presented in later chapters. We begin in Section 3.1 by defining two abstract primitives—signature schemes and zero-knowledge proof systems (as well as their realization in a noninteractive environment)—as these tools are required for both our protocol for accountable compelled access to data (Chapter 4) and our protocol for general-purpose accountable algorithms (Chapter 5. We give general definitions for the required tools and, where appropriate, the specific constructions of these primitives actually used in our implementations. We consider chapter-specific primitives in the order that the chapters are presented: Section 3.2 describes the tools required for Chapter 4 and Section 3.3 describes the tools required for Chapter 5.

3.1 Common Primitives

We begin by defining signature schemes and the concept of *zero-knowledge proof systems*, both of which are needed by our two major protocols: the protocol for accountable compelled data access presented in Chapter 4 and the more general protocol for accountable computation presented in Chapter 5.

We also cover extensions of zero-knowledge proofs to a non-interactive setting, where the proof is a single message from the prover to the verifier. Interactive zero knowledge is used in our protocol for accountable compelled data access (Chapter 4), to ensure online the honesty of participants in certain sub-protocols. Non-interactive zero knowledge lies at the core of our general purpose protocol for accountable algorithms (Chapter 5) and forms the basis for ensuring the validity of audit records for a variant of our compelled data access protocol which provides online auditability. Below, in Section 3.3.2, we discuss newly developed, efficient non-interactive zero-knowledge systems called *succinct arguments* where proofs are only computationally sound and which apply specifically to assertions about the execution of a program, a problem known as *verified computation*.

3.1.1 Signature Schemes

A *digital signature scheme* is a cryptographic tool for *authenticating* a particular string of bits. A signature scheme is a tuple of algorithms $\mathcal{S} = \{\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerifySig}\}$ where the key generation algorithm generates a private signing key and a public verification key, both uniformly at random from the space of possible keys, $(\mathsf{sk}, \mathsf{vk}) \xleftarrow{R} \mathsf{KeyGen}$; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m, r)$ is a signature generated from a secret signing key sk, a message m, and (possibly) some randomness r; and the verification algorithm accepts or rejects the signature using the verification key, $\{0,1\} \leftarrow \mathsf{VerifySig}(\mathsf{vk}, \sigma)$. Signature schemes were conjectured by Diffie and Hellman in 1976 [123], but the first concrete construction is due to Rivest, Shamir, and Adleman in 1978 [313]. We use this construction in our work; it is summarized below. Many additional signature schemes followed quickly [236, 262, 263, 303]. Goldwasser, Micali, and Rivest introduced a formal model for the security of signature schemes and presented a new scheme, the first which could be proved to meet their security requirements [176]. We turn to this model now.

The strongest security model for a signature scheme prevents *existential forgery* of signatures under an *adaptive chosen-message attack*. In the chosen-message attack model, the adversary $\mathcal{A}$ may choose messages $m_1, \ldots, m_n$ and receive corresponding signatures $\sigma_1, \ldots, \sigma_n$. The model can be made adaptive by allowing the adversary to choose message m_k after learning signature σ_{k-1}. An existential forgery is an attack where the attacker computes a valid signature $\sigma_{m'}$ on some message $m' \notin \{m_1, \ldots, m_n\}$. This is stronger than a *selective forgery* attack, in which the adversary chooses m', or a *universal forgery* attack, which results in the ability for the adversary to forge signatures on arbitrary messages. Stronger still is a total break of the signature scheme, in which the attacker learns sk.

Signatures are useful for authentication because they cannot be repudiated: the ability to generate a valid signature is, in effect, a proof of knowledge of the signing key. If the signing key is known to belong to a particular entity and that entity is trusted to hold the key securely, then any signed message from that entity must be known to originate from the entity validly. This principle underlies the use of *public key infrastructure* (PKI) for secure communication, in which certain trusted entities produce *certificates*, or digitally signed statements binding an entity name to a verification key. Verifying a signature using the verification key from a certificate, then, allows an end user to trust that the message originated from the entity named in the certificate, as long as the end user trusts the source of the signature on the certificate. Bootstrapping trust in such systems remains an important open problem in computer security.

RSA Signatures

The first known signature construction, due to Rivest, Shamir, and Adleman, still sees wide use today, including in our work in Chapters 4 and 5. We summarize it here. First, KeyGen generates distinct primes p, q and computes $N = pq$, along with

integers e, d such that $ed = 1(\mod \phi(N))$ where ϕ is the Euler totient function, or the number of integers less than its argument which are relatively prime to its argument. Set $\mathsf{vk} = (N, e)$ and $\mathsf{sk} = d$. To sign a message m, the signer computes $\sigma = m^d(\mod N)$. To verify a signature, a receiver checks that $\sigma^e = (m^d)^e = m^{de} = m^1 = m(\mod N)$, and accepts the signature if so, rejecting if not.

We observe that "plain" RSA signatures, as described above, are insecure because they are malleable: a valid signature σ_1 on a message m_1 and a valid signature σ_2 on m_2 may be multiplied to get $\sigma_3 = \sigma_1\sigma_2$, a valid signature on $m_3 = m_1m_2$. This problem can be solved by using a collision resistant hash function to produce $\eta = H(m)$ and computing signatures $\sigma = \eta^d(\mod N)$. This construction can be proved adaptive-CMA secure in the random oracle model. Other constructions exist for secure signatures without the random oracle assumption. However, we use RSA signatures on hashed messages in our work for ease of implementation.

3.1.2 Zero-Knowledge Proofs

We follow the presentation of zero-knowledge proofs in Goldreich [168]. Informally, a zero knowledge proof protocol is an interaction between a prover and one or more verifiers which allows a *prover* to demonstrate to a *verifier* that the truth of some assertion (e.g., that the prover knows the solution to a particular discrete log problem) while yielding no additional information for the verifier. Informally, receiving a zero-knowledge proof that an assertion holds is equivalent to being told that the assertion holds by a trusted party. Zero-knowledge proofs were introduced in 1985 by Goldwasser, Micali, and Rackoff [175] and have become a central feature of modern cryptography. In particular, every language L in $\mathcal{NP}$ has a zero-knowledge proof system for assertions about membership of a test string in the language. That is, for all languages $L \in \mathcal{NP}$, there exists a zero knowledge proof system for statements of the form $x \in L$. Unlike in mathematics, where proofs are considered as fixed objects

relating commonly agreed axioms to each other using commonly agreed derivation rules, zero knowledge proofs are, in general, interactive protocols played between the prover and the verifier. For this section, we discuss interactive proofs only as they relate to zero-knowledge proofs, however they are an interesting and important subject in their own right and are covered in great detail by Arora and Barak [15] and by Goldreich. In Section 3.1.2, we describe extensions that yield such protocols which consist of only a single message from the prover to the verifier—and which hence are noninteractive—but which still satisfy the other requirements of a zero-knowledge proof. Section 3.3.2 describes a particular realization of non-interactive zero knowledge proofs which are very efficient to verify, and which we use heavily in Chapter 5, namely zero-knowledge succinct arguments of knowledge, or zk-SNARKs.

In general, zero knowledge proofs are useful for partial transparency, to convince a verifier that a secret value s is well formed, has a particular structure, or satisfies some predicate, without revealing that value to the verifier. We use them for precisely this purpose in Chapters 4 and 5, to demonstrate that secret values presented to an oversight body are the same values that were used for other purposes, namely in decrypting encrypted records (Chapter 4) or in computing some decision policy (Chapter 5).

We can characterize $\mathcal{NP}$ by way of a proof system: for every language $L \in \mathcal{NP}$, L is defined by a polynomial-time recognizable relation R_L such that

$$L = \{x : \exists y \,\text{s.t.}\, (x, y) \in R_L\}$$

where $|y| \leq \text{poly}(|x|)$. Hence, y is a "proof" of the statement $x \in L$. Only correct statements of this form will have such proofs y, by definition; further, all values x in the language will have some proof y, by the definition of L. This gives us intuition

for the two main properties that are fundamental to any proof system (respectively, its verification procedure):

Soundness: The verification procedure only accepts proofs of true statements. It is impossible to provide a proof of a false statement which the verification procedure will accept.

Completeness: For any true statement, there exists a proof y for which the verification procedure will accept.

Not every collection of true statements admits a proof system with both of these properties, as was famously proved by Gödel [166], or a proof system which can be efficiently executed or even decided, as was famously proved by Turing [352]. Here, therefore, we are interested only in efficient proof systems, namely those which admit efficient discovery and verification of proofs.

Given an interactive proof system (P, V) for a language L, we say that the system is *perfect zero-knowledge* if for every probabilistic polynomial-time interactive machine V^* there exists an ordinary probabilistic polynomial-time algorithm M^* such that for every $x \in L$ the following two random variables are identically distributed:

$\langle P, V^* \rangle(x)$ (i.e., the transcript of the interaction between P and V^* on common input x)

$M^*(x)$ (i.e., the output of the machine M^* on input x.)

We say that M^* is a *perfect simulator* for the interactive protocol $\langle P, V^* \rangle$.[1] It is important to observe that this is required for *every* interacting V^*, not just the honest verifier V. If this property holds only for V, the protocol is said to be *honest-verifier*

[1] Technically, this definition is slightly too strict, in the sense that it only admits trivial protocols. We can relax the definition by allowing $M^*(x)$ to output on input x a distinguished symbol $\perp$ with probability at most 1/2. Then we define the simulation variable as m^*(x) to be the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$. That is, $\mathsf{Pr}[m^*(x) = \alpha] = \mathsf{Pr}[M^*(x) = \alpha | M^*(x) \neq \perp]$ for every $\alpha \in \{0,1\}^*$.

zero-knowledge. If we require only that the above random variables are computationally indistinguishable,[2] then we say that the protocol is *computational zero-knowledge*. A slightly stronger relaxation is to say that the distributions of the above random variable should be "statistically close", that is, negligible as a function of $|x|$. In that case, we say that a protocol is *statistical zero-knowledge*. For simplicity, we use the term "zero-knowledge" throughout this dissertation to mean "computational zero-knowledge", and will specify otherwise when necessary.

The zero-knowledge property as we have defined it is a property of the prover—it protects the prover from disclosing information to *any* probabilistic polynomial time adversarial verifier interacting with it. The simulation paradigm proves that V^* does not learn any information, as the simulator M^* does not have access to the interactive machine P, which has the secret information of interest, yet it can simulate the interaction of P with V^*.

There exists an efficient zero-knowledge proof system for any language in $\mathcal{NP}$, as can be seen by constructing one for any $\mathcal{NP}$-complete language. We give here a construction of a proof system for graph 3-colorability, an arbitrarily chosen $\mathcal{NP}$-complete language. The language of 3-colorable graphs consists of all simple finite graphs (i.e., no parallel edges or self-loops, where an edge has both ends on a single vertex) that can be *vertex-colored* using three colors such that no two adjacent vertices are assigned the same color. Formally, a graph $G = (V, E)$ is three-colorable if there exists a map $\phi : V \longrightarrow \{1, 2, 3\}$ such that $\phi(u) \neq \phi(v)$ for all $(u, v) \in E$.

An interactive proof that a prover knows a 3-coloring of G is as follows. The prover and verifier receive as common input a graph $G = (V, E)$ with $V = \{1, \ldots n\}$. The prover receives as auxiliary input a 3-coloring ϕ.

[2] That is, for every probabilistic polynomial time adversary $\mathcal{A}$, the probability $|\mathsf{Pr}[\mathcal{A}(\langle P, V^*\rangle(x)) = 1] - \mathsf{Pr}[\mathcal{A}(M^*(x)) = 1]| \leq \epsilon(k)$, where the probability is taken over the choice of x and the randomness of $\mathcal{A}$, M^*, and the interactive protocol $\langle P, V^*\rangle$. We denote by $\epsilon(k)$ a negligible function of a security parameter k.

The prover selects a random permutation π over $\{1, 2, 3\}$ and sets $\psi(v) := \pi(\phi(v))$ for all $v \in V$. The prover commits (see Section 3.3.1) to the color of each vertex independently, computing independently $(C_{\psi(v)}, r_{\psi(v)}) = \mathsf{Commit}\,[\psi(v)]$ for each $v \in V$ and sending all commitments $C_{\psi(1)}, \ldots C_{\psi(n)}$ to the verifier.

The verifier uniformly selects an edge $(u, v) \in E$ and sends it to the prover.

The prover opens the commitments for vertices u and v.

The verifier verifies the opening of the commitments it has received and confirms that these colors are different. If so, the verifier accepts. If not, it rejects.

This protocol is a valid zero-knowledge proof for 3-colorability, as can be seen by observing that the verifier will always accept if the prover knows a valid 3-coloring. If the prover's auxiliary input is *not* a valid 3-coloring of G, then some edge $(u, v) \in E$ will have $\phi(u) = \phi(v)$ and hence $\psi(u) = \psi(v)$. Thus, the verifier has a nontrivial probability of detecting cheating by the prover. This probability may be amplified to the verifier's desired level of confidence by running the protocol repeatedly (because a cheating prover would need to cheat in *all* instances of the protocol, a detection probability of p would imply an overall confidence of $(1 - p^n)$ if the protocol were run n times).

The protocol is zero-knowledge, as we can construct a simulator. Given a program V^* that interacts with the prover, we construct M^* to first produce a "pseduo-coloring" of G by selecting uniformly values $e_1, \ldots e_n \in \{1, 2, 3\}$ and committing to each of them. This sequence of commitments is computationally indistinguishable to a sequence of commitments from a real interaction with P. If V^* asks to examine an edge $(u, v) \in E$ for which $e_u \neq e_v$, then M^* can open the corresponding commitment and answer correctly. If, however, V^* asks to examine an edge for which $e_u = e_v$, M^* has no way to answer correctly and so aborts the simulation, outputting the distinguished symbol $\perp$. This happens with probability $1/3$, and so M^* does not

output $\perp$ with probability 2/3. This is consistent with our (relaxed) definition of zero-knowledge.

This shows that there exist computational zero-knowledge interactive proof systems for all languages in $\mathcal{NP}$. However, to accomplish this, we have had to assume the existence of *one-way functions* in order to achieve the commitments necessary in our protocol. This requirement is thought to be inherent. Below, Section 3.3.2 describes systems with a different profile of properties, which are better suited to our needs in later chapters. Goldreich gives an excellent overview of results and theory in this area [168].

Non-Interactive Zero Knowledge

The above-described protocol for zero-knowledge requires interaction between the prover and the verifier. Blum, Feldman, and Micali introduced a variant of zero-knowledge proofs in which the interaction between the prover and the verifier consists of a single message from the prover to the verifier, and so no actual interaction is necessary [56]. Specifically, Blum et al. showed that a common reference string shared between the prover and verifier is sufficient to achieve zero-knowledge without interaction. Such a string could be generated during a special setup phase, and in some applications has special structure or must be known to have been sampled uniformly from some space. Subsequently, Goldreich and Oren showed that even one-shot zero-knowledge proofs are impossible in the standard model [171], meaning that some additional assumption is necessary to achieve them. Fiat and Shamir showed that the common random string can be generated heuristically from what would be the verifier's view of an interactive protocol, though the use of random oracles [148]. Later authors generalized and improved the concept substantially [55,142,143,187,329]. We consider non-interactive zero-knowledge further in Section 3.3.2.

3.2 Primitives Required for Accountable Warrant Execution

We review here the particular primitives necessary for the protocol for accountable compelled access to data described in Chapter 4. These are: (i.) identity-based encryption (Section 3.2.1), (ii.) Oblivious Transfer (Section 3.2.2), (iii.) secret sharing schemes (Section 3.2.3), and (iv.) threshold cryptography (Section 3.2.4).

3.2.1 Identity-Based Encryption

Identity-based cryptography defines public-key cryptosystems (encryption and signature schemes) where the public keys are arbitrarily chosen strings called *identities*, intended to be non-repudiable names or addresses of the participants.[3] In identity-based cryptography, all users must trust a central key generation authority, which holds long-term secrets used to derive an identity-specific secret key for each user. The key authority need not be online once identity keys have been generated and does not participate directly in protocols using identity-based primitives. The goal of designing systems in this way is to eliminate the need for any sort of key distribution mechanism such as public key infrastructure (PKI) [209], a "web of trust" consisting of signed key endorsements shared among participants [157, 377], or the trust of a third-party key distribution service [35, 341] with which each communicating party must maintain a shared long-term secret.

Shamir introduced the concept of identity-based cryptosystems over 30 years ago, in 1984 [332]. However, Shamir was only able to construct identity-based signatures. Constructing identity-based encryption (IBE) remained an open problem until 2001,

[3] In this way, identity-based cryptography shares some goals with the design of accountable systems, as it is intended as a tool for attributing cryptographic operations. Shamir said in his original paper on the topic that keys could be "[a]ny combination of name, social security number, street address, office number, or telephone number [...] provided that it uniquely identifies the user in a way he cannot later deny, and that is readily available to the other party."

when two constructions were discovered: one by Boneh and Franklin [63, 64] using *pairings*, or bilinear maps between groups, and one by Cocks [100] based on quadratic residues. Later IBE schemes focused on the pairing-based approach [59, 60, 324, 361], as it is more practical. A pairing is an efficiently computable (non-trivial) bilinear mapping $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ where $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are finite cyclic groups of some prime order q. We let g be a generator of $\mathbb{G}$ and $\hat{g}$ be a generator of $\hat{\mathbb{G}}$. A method for publicly verifiable parameter generation for typical instantiations of pairing-based systems was developed by Chen, Cheng, and Smart [89].

Briefly, an IBE system consists of four algorithms:

Setup$_{\mathbf{IBE}}$ generates a master public key mpk and corresponding master secret key msk;

Extract$_{\mathbf{IBE}}$ uses the master secret key msk and an identity id to generate a corresponding secret key $\mathsf{sk}_{\mathsf{id}}$;

Enc$_{\mathbf{IBE}}$ encrypts a message m using id, yielding $\mathsf{ct}_{\mathsf{id}}$; and

Dec$_{\mathbf{IBE}}$ decrypts a ciphertext $\mathsf{ct}_{\mathsf{id}}$ using $\mathsf{sk}_{\mathsf{id}}$.

IBE Security requires that an adversary who learns the secret keys for multiple identities cannot break the security of the scheme for some other identity id for which the adversary does not have $\mathsf{sk}_{\mathsf{id}}$. The IBE we are interested in, that of Boneh and Boyen and described in Section 3.2.1, satisfies a slightly weaker *Selective* IBE security property, in which the adversary must commit to attacking a particular id before learning any identity keys, rather than making this choice adaptively; BB-IBE-like schemes can be made adaptive-ID secure [361], but selective security suffices in our work.

Boneh-Boyen IBE

The protocol of Chapter 4 makes use specifically of the Boneh-Boyen IBE (BB-IBE) scheme [59, 60]. We describe the operation of this scheme briefly:

$\mathbf{Setup}_{\mathbf{IBE}}$ chooses a random generator $g \in \mathbb{G}$, a random $\alpha \in \mathbb{Z}_q$, and random elements $h, g_2 \in \mathbb{G}$. Letting $g_1 := g^\alpha$ and $v := e(g_1, g_2)$, the algorithm outputs:

$$\mathsf{mpk} := (g, g_1, h, v) \quad \text{and} \quad \mathsf{msk} := g_2^\alpha$$

$\mathbf{Extract}_{\mathbf{IBE}}(\mathsf{msk}, \mathbf{id})$ chooses a random $r \in \mathbb{Z}_q$ and outputs:

$$\mathsf{sk}_{\mathsf{id}} := (d_0, d_1) = \left(\mathsf{msk} \cdot \left(g_1^{\mathsf{id}} h\right)^r,\ g^r \right)$$

$\mathbf{Enc}_{\mathbf{IBE}}(\mathsf{mpk}, \mathbf{id}, m)$ encrypts $m \in \hat{\mathbb{G}}$ by choosing a random $s \in \mathbb{Z}_q$ and producing:

$$\mathsf{ct} := \left(\left(v^s \cdot m,\ g^s,\ (g_1^{\mathsf{id}} h)^s\right) \right)$$

$\mathbf{Dec}_{\mathbf{IBE}}(\mathsf{sk}_{\mathbf{id}}, \mathsf{ct})$ considers ct as the tuple (A, B, C) and outputs:

$$A \cdot \frac{e(C, d_1)}{e(B, d_0)} = e(g_1, g_2)^s \cdot m \cdot \frac{e(g_1^{\mathsf{id}} h, g)^{rs}}{e(g, g_2)^{s\alpha} \cdot e(g, g_1^{\mathsf{id}} h)^{rs}} = e(g_1, g_2)^s \cdot m \cdot \frac{1}{e(g, g_2)^{s\alpha}} = m$$

where in the last equality, we have used the fact that $g_1 := g^\alpha$.

The exact operation of $\mathsf{Dec}_{\mathrm{IBE}}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$ is not relevant to our discussion in Chapter 4—it is included for completeness only. Selective IBE security follows from a standard assumption on pairing friendly elliptic curves. BB-IBE encryption does not use pairings and is comparable in speed to a standard elliptic-curve public-key encryption. As far as we are aware, the protocol introduced in Chapter 4 is the first to benefit substantially from this feature. By contrast, other IBEs, such as those of Boneh

and Franklin [63] or Sakai and Kasahara [324] require a pairing computation during encryption.

3.2.2 Oblivious Transfer

Oblivious transfer has an extensive history in the literature [78, 139, 182, 183, 211, 213, 279, 304]. At a high level, oblivious transfer is an interactive protocol between two parties, a *sender*, $\mathcal{S}$, and a *receiver*, $\mathcal{R}$ where:

- The sender has a set of messages $S = \{m_1, \ldots, m_s\}$.
- The receiver takes as input an index $\ell \in \{1, \ldots, s\}$.

After the protocol, $\mathcal{R}$ receives message m_ℓ and nothing else, while $\mathcal{S}$ learns nothing. Modern oblivious transfer protocols allow the receiver to adaptively query for multiple messages.

The first oblivious transfer protocol is due to Rabin [304]. This protocol proceeds as follows:

Protocol (Rabin's Oblivious Transfer).

(i.) $\mathcal{S}$ picks a value $N = pq$, where p and q are large primes, and a value e relatively prime to N. $\mathcal{S}$ encrypts a message m as $c \equiv m^e \mod N$ and sends N, e, c to $\mathcal{R}$.

(ii.) $\mathcal{R}$ chooses at random $x \mod N$ and sends a to $\mathcal{S}$, where $a \equiv x^2 \mod N$.

(iii.) $\mathcal{S}$ computes y such that $y^2 \equiv a \mod N$ and returns y to $\mathcal{R}$.

If $y \in \{x, -x\}$, $\mathcal{R}$ has learned nothing. If, however, $y \notin \{x, -x\}$, $\mathcal{R}$ will have learned enough to efficiently factor N and recover m from the known values e and $c \equiv m^e \mod N$. Because $\mathcal{S}$ cannot learn x from a as it is Rabin-encrypted [303], $\mathcal{S}$ does not know whether y allowed $\mathcal{R}$ to recover m or not. In this way, $\mathcal{S}$ is oblivious as

to whether the message was sent. Rabin's approach was later generalized to a method for all or nothing disclosure of s secrets by Brassard, Crépeau, and Robert [71].

A later protocol of Even, Goldreich, and Lempel [139], attributed by the authors also to Micali, defines a 1-out-of-2 oblivious transfer where $\mathcal{S}$ has $S = \{m_0, m_1\}$ and $\mathcal{R}$ wishes to receive m_b for $b \in \{0, 1\}$ while holding $\mathcal{S}$ oblivious to the value of b. This was later generalized to 1-of-s oblivious transfer [279] and to transfer of arbitrary subsets from a permissible collection [211].

These two definitions (that of Even, Goldreich, and Lempel in terms of indexing into a set and that of Rabin in terms of all-or-nothing disclosure) were shown to be equivalent by Crépeau [107].

Blind IBE and Adaptive Oblivious Transfer

Green and Hohenberger [182] define an extension to IBE called *blind IBE*, in which the protocol $\mathsf{Extract}_{\mathrm{IBE}}$ used by message recipients to receive identity keys $\mathsf{sk}_{\mathsf{id}}$ for their identity id from the IBE key authority is replaced by a new protocol $\mathsf{BlindExtract}_{\mathrm{IBE}}$ in which message recipients receive $\mathsf{sk}_{\mathsf{id}}$ but the key authority learns nothing about the id for which it produces $\mathsf{sk}_{\mathsf{id}}$. This can be thought of as an oblivious transfer of $\mathsf{sk}_{\mathsf{id}}$ from the key authority acting as $\mathcal{S}$ to a user acting as $\mathcal{R}$, where the messages $\mathsf{sk}_{\mathsf{id}}$ are indexed by identities id in a set of possible identities $\mathcal{I}$.

Indeed, Green and Hohenberger show that blind IBE yields an efficient adaptive oblivious transfer where messages $S = \{m_1, \ldots, m_s\}$, drawn from a universe of possible messages $S \subseteq \mathcal{M}$, are IBE encrypted such that m_i is encrypted under the identity i. The receiver, wishing to receive message m_σ, where $\sigma \in \{1, \ldots, s\}$, blindly extracts $\mathsf{sk}_{\mathsf{id}}$ for $\mathsf{id} = \sigma$ and then decrypts m_σ as desired.

This simple protocol can be extended to an adaptive oblivious transfer that is *fully simulation secure*, meaning that security for both the sender and the receiver can be proved by showing that neither party can determine whether they are participating

in the real protocol or in an ideal-world protocol where the other party is assumed trusted.[4] Green and Hohenberger further extended this to a protocol that is *universally composable* [183], meaning that it is secure even when composed arbitrarily with other protocols or executed concurrently [79].

We describe here the concrete blind IBE Green and Hohenberger derive for BB-IBE, as we will need it in Chapter 4. The protocol is between the IBE key authority $\mathcal{P}$ and a user $\mathcal{U}$.

Protocol (Blind BB-IBE Extraction).

$\mathcal{U}$:

(i.) Choose $y \xleftarrow{R} \mathbb{Z}_q$.

(ii.) Compute $h' = g^y g_1^{\mathsf{id}}$ and send h' to $\mathcal{P}$.

(iii.) Execute $PoK\{(y, \mathsf{id}) : h' = g^y g_1^{\mathsf{id}}\}$ with $\mathcal{P}$.[5]

$\mathcal{P}$:

(iv.) Abort if the proof fails to verify.

(v.) Choose $r \xleftarrow{R} \mathbb{Z}_q$.

(vi.) Compute $d_0' = \mathsf{msk} \cdot (h'h)^r$.

(vii.) Compute $d_1' = g^r$.

(viii.) Send (d_0', d_1') to $\mathcal{U}$.

$\mathcal{U}$:

(ix.) Check that $e(g_1, g_2) \cdot e(d_1', h'h) = e(d_0', g)$.

[4] Many oblivious transfer protocols were introduced in a "half simulation" model, meaning that only the security of the sender was proved using a real/ideal simulation game and only a weaker notion of security can be proved for the receiver [125, 277, 278, 280, 288]. However, Naor and Pinkas showed that this model admits practical "selective failure" attacks on the receiver's privacy—the sender can introduce protocol failures that depend on the receiver's choice of message [278].

[5] This can be done easily using the techniques of Schnorr [329].

(x.) If the check passes, choose $z \xleftarrow{R} \mathbb{Z}_q$; otherwise, abort and output $\perp$.

(xi.) Compute $d_0 = (d'_0/(d'_1)^y) \cdot (g_1^{\mathsf{id}} h)^z$ and $d_1 = d'_1 \cdot g^z$.

(xii.) Output $\mathsf{sk}_{\mathsf{id}} = (d_0, d_1)$.

Green and Hohenberger prove that this protocol is secure in the sense that:

(i.) It is *leak-free*, meaning that a malicious user cannot learn anything by executing $\mathsf{BlindExtract}_{\mathrm{IBE}}$ with an honest authority that she could not learn by executing $\mathsf{Extract}_{\mathrm{IBE}}$ with an honest authority. Further, the user must actually know the id for which she is extracting a key.

(ii.) It is *selective failure blind*, meaning that a malicious authority cannot learn about the identity for which a user has requested the extraction of a secret key, nor can the authority cause the $\mathsf{BlindExtract}_{\mathrm{IBE}}$ protocol to fail in a manner that depends on the user's choice of identity.

3.2.3 Secret Sharing

Secret sharing (also called *secret splitting*) refers to any of several approaches to distributing a secret value among a group of participants, each of whom holds a *share* of the secret. The secret can only be reconstructed if a sufficient number of shares are combined. The shares may be distributed by a trusted *dealer*, or the secret may be jointly constructed by the players' choice of individual shares [32].[6] The concept of secret sharing was independently introduced in 1979 by Shamir [331] and Blakley [51].

Formally, let s be a secret, drawn from some universe of possible secrets $s \in \mathcal{S}$. The goal is to create a set H of shares of s, $H = \{s_1 \ldots s_N\}$ such that: (i.) Knowledge

[6]This latter approach is useful if the secret to be shared may be chosen at random, as cryptographic keys generally are. Much work has described distributed protocols to generate secrets shared in this way which nonetheless have some algebraic structure required for certain cryptographic applications. These protocols, called "distributed key generation" (DKG) protocols in the literature, often have the added advantage that, so long as one participant plays honestly, the secret determined by the protocol is randomly sampled in the sense that it is drawn uniformly from the space of possible secrets and is unpredictable to all parties [40, 82, 89, 163].

of a subset $T \subseteq H$ where $|T| \geq t$ implies knowledge of s; and (ii.) Knowledge of a subset $T \subseteq H$ where $|T| < t$ imparts no additional knowledge of s (i.e., an adversary who learns T has an unchanged prior distribution over the values in $\mathcal{S}$ for the value of s).[7] Such a scheme is referred to as a (t, N)-secret sharing scheme and the set H is referred to as a *sharing* of s. If $t = N$, all participants are required to reconstruct the secret. Typically, the secret may be reconstructed by any subset of $\geq t$ of the N participants, but this varies based on the particulars of the scheme in use. The cases of $t = 1$ and $t = N$ are trivial to construct schemes for—when $t = 1$, the secret may be given to all participants; when $t = N$, all parties except one may be given random bit strings equal in length to the secret and the last party is given the bitwise `XOR` of these strings with the secret. The cases $1 < t < N$ are interesting, however, as are subset-oriented access control rules, which can be simulated by giving certain participants multiple shares (as an easy example, if the president of a company should always be able to access the secret, he or she can be given t shares in a scheme with $t + N$ total shares).

Shamir and Blakley both proposed making the secret a kind of linear reconstruction of algebraic information—Shamir proposed encoding the secret as the constant term of a degree $t-1$ polynomial and giving each player a point on the graph of that polynomial, while Blakley proposed encoding the secret as a coordinate at the intersection of t $(t-1)$-dimensional hyperplanes. Other early schemes due to Mignotte and Asmuth and Bloom used the Chinese Remainder Theorem [17, 266]. Later, Chor et al. introduced the concept of *verifiable secret sharing* [92], in which participants can check that their shares are consistent (i.e., are shares of the same secret), which protects them from a misbehaving dealer and from malicious protocol participants

[7] Technically, this definition only admits secret sharing schemes which are information-theoretically secure (because an adversary with $< t$ shares has no additional information on the secret). Information-theoretically secure schemes require each participant to hold an amount of information equal to $|s|$, and therefore have a total storage cost of $N|s|$. To improve on this, some secret sharing schemes in the literature are only computationally secure, but we do not describe them here.

trying to learn information beyond their own shares. Pedersen introduced a publicly verifiable scheme, which allows anyone to verify the consistency and validity of shares [296]. Secret sharing is an important primitive for many secure multiparty computation protocols; a survey of secret sharing schemes and their uses can be found in Beimel [32].

Shamir Secret Sharing

We describe the secret sharing scheme of Shamir [331] in more detail, as it is necessary to construct threshold BB-IBE, described in Section 3.2.4.

To construct a (t, N)-secret sharing, Shamir observes that a polynomial $f(x)$ of degree $k-1$ is fully determined by any k points that lie on its graph while any value of f is consistent with knowledge of only $k-1$ points; hence, anyone who knows at least k points can interpolate and discover the full polynomial while anyone knowing only $k-1$ points will have no information about the polynomial f. Therefore, assuming the secret s is an element of some finite field $\mathbb{F}$, we construct a degree-$(t-1)$ polynomial over $\mathbb{F}$ by choosing at random $a_1, \ldots, a_{t-1}$, setting $a_0 = s$, and letting:

$$f(x) := a_0 + a_1 x + \ldots a_{t-1} x^{t-1}$$

Now we can construct a sharing of s for all N participants simply by evaluating $s_i := f(i)$.

3.2.4 Threshold Cryptography

In general, if a secret key operation in a cryptosystem is performed by a group of N participants working in concert, and that operation requires at least a threshold t of the participants to participate in order to be successful, that cryptosystem is said to be a (t, N)-*threshold* cryptosystem. Practically, threshold cryptosystems may be

constructed whenever secret key operations in a certain system commute (or can be made to commute by altering the protocol) with a secret sharing scheme. Threshold cryptography was introduced by Desmedt [119] and built on by Boyd [67], Croft and Harris [108], and Desmedt and Frankel [120]. Threshold algorithms exist for many important public-key cryptosystems, including RSA [335], ElGamal [120], Paillier [151], and the digital signature algorithm [167].

Blind Threshold BB-IBE

We use a threshold blind identity key extraction protocol for BB-IBE in Chapter 4. This protocol allows for a configurable threshold t of N total decryption authorities to participate in extracting identity keys by using Shamir's secret sharing on the master secret key msk. Specifically, during the setup phase, each player is dealt $\mathsf{msk}_i = g_2^{\alpha_i}$ where each $(\alpha_1, \ldots, \alpha_n)$ is a linear sharing of the original α in $\mathsf{msk} = g_2^{\alpha}$.[8]

During $\mathsf{BlindExtract}_{\mathrm{IBE}}$, we follow the protocol described in Section 3.2.2, although now the protocol is executed between a user $\mathcal{U}$ and some active subset $W \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ of the N distributed key authorities. Thus, in step (ii.), the user sends h' to the set of active parties W. Once $\mathcal{U}$ has amassed $\geq t$ blinded key shares $\mathsf{sk}_{\mathsf{id},i} = (d_{0,i}, d_{1,i})$, she can combine them by choosing $z \xleftarrow{R} \mathbb{Z}_q$ and computing:

$$\mathsf{sk}_{\mathsf{id}} = \Big(\prod_{i \in W} \frac{d_{0,i}^{\lambda_i}}{d_{1,i}} (g_1^{\mathsf{id}} h)^z, \prod_{i \in W} (d_{1,i})^{\lambda_i} g^z \Big)$$

Where the λ_i are the secret sharing coefficients, that is, $\alpha = \sum_{i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}} \lambda_i \alpha_i$, where the values of the λ_i are determined by interpolating to recover the secret, as in the Shamir scheme.[9]

[8] These keys (and their sharing) can be generated by a trusted dealer or constructed using the distributed, multi-party key generation protocols of Gennaro et al. [163].

[9] Technically, any linear secret sharing scheme would work, but we use Shamir secret sharing in our implementation of the protocol in Chapter 4.

3.3 Primitives Required for General-Purpose Accountable Algorithms

We begin by considering the tools necessary for our general-purpose protocol for building accountable algorithms. There are four basic tools:

(i.) cryptographic commitments (Section 3.3.1), (ii.) zero knowledge and non-interactive zero-knowledge proofs with a special focus on zk-SNARKs (Section 3.1.2), (iii.) pseudorandomness (Section 3.3.3), and (iv.) techniques for obtaining fair random bits (Section 3.3.4).

3.3.1 Cryptographic Commitments

Cryptographic commitments are a basic ingredient in many protocols and were introduced in 1979 by Shamir, Rivest, and Adleman [333] and separately by Blum in 1981 [54], and Even, Goldreich, and Lempel in 1985 [139]. Later authors formalized the notion more carefully and introduced the alternative terminology of *bit commitment schemes*, used interchangeably with the plain term "commitment", but sometimes reserved for the special case where the value being committed to is a single bit [70,276]. At a high level, a commitment scheme is a *two-phase* two-party protocol in which one party, called the *sender* $\mathcal{S}$, publishes a value C_x during the first phase (the *commit phase*) which binds it to a particular *value* x. During the second phase (the *reveal phase*), the commitment is *opened* to reveal x to the receiver $\mathcal{R}$ such that the following requirements are satisfied.

Hiding: At the end of the first phase, $\mathcal{R}$, having learned C_x, learns nothing about x.

Binding: Given the transcript of the first phase, there exists only a single value that the receiver can accept in the second phase as a valid *opening* of the commitment. This requirement must hold even if $\mathcal{S}$ tries to cheat.

More formally, a commitment scheme is a pair of randomized algorithms $\mathsf{Commit}\,[x] \rightarrow (C_x, r_x)$ and $\mathsf{VerifyCommitment}\,[x, C_x, r_x] \rightarrow \{0,1\}$ with the following properties:

$\mathsf{Commit}\,[x] \rightarrow (C_x, r_x)$: The commitment algorithm takes a value x and yields a tuple (C_x, r_x) of a *commitment* and a *commitment key*. The commitment must preserve secrecy in the sense of the hiding property above, namely that C_x may be safely published without leaking any information about x.[10] The commitment key r_x is a random value drawn from $\{0,1\}^k$, where k is a security parameter for the commitment scheme.

$\mathsf{VerifyCommitment}\,[x, C_x, r_x] \rightarrow \{0,1\}$: The commitment verification algorithm takes the revealed previously committed value x, the commitment C_x, and the commitment randomness r_x and returns a single bit, 1 if $\mathsf{Commit}\,[x]$ could have yielded (C_x, r_x) for some choice of its internal random coins and 0 otherwise. The probability that a computationally bounded adversary can determine values (x, x', C_x, r_x) such that $(x, r_x) \neq (x', r'_x)$ and $\mathsf{VerifyCommitment}\,[x, C_x, r_x] = 1$ but also $\mathsf{VerifyCommitment}\,[x', C_x, r'_x]$ is a negligible function of k.

Random Oracle Commitments

Commitment schemes can be realized using any one-to-one one-way function and its hard-core predicate [168]. We realize commitments in the random oracle model using a cryptographic hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^k$. Given a security parameter κ, to commit to a value x, we select uniformly $r \stackrel{R}{\leftarrow} \{0,1\}^\kappa$ and let $C_x = H(x \| r)$.

$$(C_x, r) \leftarrow \mathsf{Commit}\,[x]$$

[10] Formally, for all x_0 and x_1, and given a bit $b \stackrel{R}{\leftarrow} \{0,1\}$, a computationally bounded adversary who sees only x_0, x_1, and C_{x_b} has negligible advantage in guessing the value of b.

The specific hash function we use, which must be efficiently realizable in a quadratic arithmetic program for computation inside a zk-SNARK (see Section 3.3.2), is described below.

An Arithmetic Hash Function

Ajtai demonstrated that certain problems related to finding short vectors in lattices are *hard on average* if, as is commonly believed, they are hard to *approximate in the worst case* [9]. Goldreich, Goldwasser and Halevi (GGH) showed that this result implies a straightforward families of hash function on m-bit strings that are both *universal* and *collision-free*. Because these constructions involve only simple arithmetic operations, they lead to compact and efficient representations in the language of *rank-1 constraint systems*, described below in Section 3.3.2, which admit efficient zero-knowledge succinct arguments, or zk-SNARKs.

The GGH hash function (on m-bit input and $n \log q$-bit output, $h_{GGH} : \{0,1\}^m \to \mathbb{Z}_q^n$) is constructed as follows: let q be a prime such that $n \log q < m < \frac{q}{2n^4}$. Then pick a random $n \times m$ matrix M with entries in $\mathbb{Z}_q$. Given $M \in \mathbb{Z}_q^{n \times m}$, we can define a function $h_M : \{0,1\}^m \longrightarrow \mathbb{Z}_q^n$, defined for input $s = s_1 s_2 \cdots s_m \in \{0,1\}^m$, as

$$h_M(s) = Ms(\mod q) = \sum_i s_i M_i(\mod q)$$

where M_i is the i^{th} column of M.

h_M maps m bits to $n \log q$ bits, but the parameters are chosen so that $m > n \log q$. This implies that there are collisions in the output of h_M. Goldreich, Goldwasser, and Halevi showed that it is infeasible to find any of these collisions unless certain well-known lattice problems have good, efficiently discoverable approximate solutions in the worst case, which would contradict the result of Ajtai mentioned above.

3.3.2 Verified Computation

We covered verified computation and its goals at a high level in Chapter 2, Section 2.1.3. Here, we discuss specific protocols as they relate to our work in later chapters. Specifically, we examine the history and operation of zero-knowledge succinct argument systems (zk-SNARKs).

Before defining a concrete zk-SNARK protocol, however, we define verified computation more formally. A zero-knowledge verifiable computation scheme $\mathcal{VC}$ for a function F consists of three polynomial-time algorithms, $\mathcal{VC} = \{\mathsf{KeyGen}, \mathsf{Compute}, \mathsf{Verify}\}$ defined as follows [291, Def. 1].

$(\mathrm{EK}_F, \mathrm{VK}_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$**:** The randomized algorithm KeyGen takes the function F to be executed and a security parameter λ; it outputs a public *evaluation key* EK_F and a public *verification key* VK_F.

$(z, \pi_z) \leftarrow \mathsf{Compute}(\mathrm{EK}_F, x, y)$**:** The deterministic worker algorithm uses the public evaluation key EK_F the input x, and the auxiliary input y. It outputs $z \leftarrow F(x, y)$ and a proof π_z of z's correctness.

$\{0, 1\} \leftarrow \mathsf{Verify}(\mathrm{VK}_F, x, \pi_z)$**:** Given the verification key VK_F and the input x, the deterministic verification algorithm outputs 1 if $F(x, y) = z$ and 0 otherwise.

Such a protocol might satisfy several desirable properties, which we summarize here. The interested reader should consult full formal definitions in the supporting literature [93, 161, 162, 291, 292]:

Correctness For any function F and input (x, y) to F, if we run $(\mathrm{EK}_F, \mathrm{VK}_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$ and $(z, \pi_z) \leftarrow \mathsf{Compute}(\mathrm{EK}_F, x, y)$, we always get $1 \leftarrow \mathsf{Verify}(\mathrm{VK}_F, x, \pi_z)$.

Security For any function F and any probabilistic polynomial-time adversary $\mathcal{A}$, $\mathsf{Pr}[((x', y'), z', \pi'_{z'}) \leftarrow \mathcal{A}(\mathrm{EK}_F, \mathrm{VK}_F) : F(x, y) \neq z \text{ and } 1 \leftarrow \mathsf{Verify}(\mathrm{VK}_F, x', z', \pi'_{\hat{\mathbf{z}}'_i}] \leq \mathrm{negl}(\lambda)$.

Efficiency KeyGen is assumed to be a one-time operation whose cost is amortized over many calculations; we require that Verifyis cheaper than simply executing F.

Zero-Knowledge After receiving π_z, no probabilistic polynomial-time verifier has learned any additional knowledge about the prover's auxiliary input y.

Computationally Sound Arguments

Zero-knowledge succinct arguments of knowledge (zk-SNARKs) realize practically the goals of zero-knowledge proof systems and verified computation systems, both described above, but as *computationally sound arguments*. This relaxes the soundness requirement of zero knowledge proofs, such that instead of being impossible for the verifier to accept a proof of a false statement, it is merely *computationally infeasible*. In exchange, it is known that perfect zero-knowledge argument systems exist for all languages in $\mathcal{NP}$ (recall that we were only able to show the existence of computational zero knowledge systems when we required perfect soundness, see Section 3.1.2. However, we cannot merely modify the usual definition of a zero-knowledge proof system such that we only require computational soundness, since the definition treats the prover as being computationally unbounded. Hence, we have to relax the completeness condition as well.

Definition 3 (Computationally Sound Arguments (Goldreich, Def. 4.8.1 [168]))**.** *A pair of interactive machines* (P, V) *is called an* argument system (computationally sound proof system) for a language L *if both machines are polynomial-time (with auxiliary inputs) and the following two conditions hold:*

Completeness *For every $x \in L$, there exists a string y such that for every string z,*

$$\mathsf{Pr}[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$$

Computational Soundness *For every polynomial-time interactive machine B, and for all sufficiently long $x \notin L$ and every y and z,*

$$\mathsf{Pr}[\langle B(y), V(x)\rangle(x) = 1] \leq \frac{1}{3}$$

As before, the confidence such a protocol provides the verifier can be improved by running the protocol many times, although parallel runs may not provide any increased confidence in certain cases.

Zero Knowledge Succinct Arguments (zk-SNARKs)

Surprisingly, argument systems can be built for any language in $\mathcal{NP}$ in a way that is *succinct*, meaning that argument messages are very small and can be verified very efficiently. Many authors have described various zk-SNARK constructions [39, 42, 50, 72, 162, 186, 187, 249, 250, 291]. We give here an overview of the construction used in our work, which has been particularly well-studied and has been carefully developed and optimized [39, 42, 162, 291]. We follow the compact presentation of this protocol in Ben-Sasson et al. [42].

Public Parameters: A prime r, two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order r with generators $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ (where $\mathbb{G}_T$ is also of order r).

Key Generation $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(C)$**:** On the input of a circuit $C : \mathbb{F}_r^n \times \mathbb{F}_r^h \longrightarrow \mathbb{F}_r^l$, the key generator G computes $(\vec{A}, \vec{B}, \vec{C}, Z) = \mathsf{QAPinst}(C)$ where $\mathsf{QAPinst}$ is a function converting circuits into *quadratic arithmetic programs*. $\mathsf{QAPinst}$ is

an involved process, and we do not describe it in full here. Instead, we direct the interested reader to Gennaro et al. [162] and remark that quadratic arithmetic programs are constraint programs over the language of rank-1 constraint systems (R1CS), where the constraints take the form:

$$\langle \vec{A}, (1, \vec{X}\rangle \cdot \langle \vec{B}, (1, \vec{X})\rangle = \langle \vec{C}, (1, \vec{X})\rangle$$

Here, $\vec{A}, \vec{B}, \vec{C}$ are vectors over a field $\mathbb{F}_r$, which coincides with the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and $\vec{X}$ is a vector of formal variables for the constraint system. We remark that there is a companion function, $S \leftarrow \mathsf{QAPwit}(C)$, which produces a function $S : \mathbb{F}_r^n \times \mathbb{F}_r^h \longrightarrow \mathbb{F}_r^\ell$ which takes inputs $(\vec{x}, \vec{y})$ to C and produces a satisfying assignment to the QAP. We also remark that the process of producing these constraint representations constitutes the bulk of programmer effort and the bulk of computational inefficiency in using zk-SNARKs in practice (the R1CS representation, being easily relatable to a circuit representation, is much larger and more unwieldy than representations of the computation in more traditional languages).

The key generator extends the QAP by adding constraints

$$\begin{aligned} A_{m+1} &= B_{m+2} = C_{m+3} = Z, \\ A_{m+2} &= A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0. \end{aligned}$$

G then samples uniformly $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma \in \mathbb{F}_r$ and sets $\mathsf{pk} := (C, \mathsf{pk}_A, \mathsf{pk}'_A, \mathsf{pk}_B, \mathsf{pk}'_B, \mathsf{pk}_C, \mathsf{pk}'_C, \mathsf{pk}_K, \mathsf{pk}_H)$ where for $i = 0, 1, \ldots, m+3$:

$$\begin{aligned}
\mathsf{pk}_{A,i} &:= A_i(\tau)\rho_A\mathcal{P}_1, \quad \mathsf{pk}'_A := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1 \\
\mathsf{pk}_{A,i} &:= A_i(\tau)\rho_A\mathcal{P}_1, \quad \mathsf{pk}'_A := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1 \\
\mathsf{pk}_{A,i} &:= A_i(\tau)\rho_A\mathcal{P}_1, \quad \mathsf{pk}'_A := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1 \\
\mathsf{pk}_{A,i} &:= A_i(\tau)\rho_A\mathcal{P}_1, \quad \mathsf{pk}'_A := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1
\end{aligned}$$

The key generator also computes $\mathsf{vk} := (\mathsf{vk}_A, \mathsf{vk}_B, \mathsf{vk}_C, \mathsf{vk}_\gamma, \mathsf{vk}^1_{\beta\gamma}, \mathsf{vk}^2_{\beta\gamma}, \mathsf{vk}_Z, \mathsf{vk}_{\mathsf{IC}}$ where

$$\begin{aligned}
\mathsf{vk}_A &:= \alpha_A\mathcal{P}_2, & \mathsf{vk}_B &:= \alpha_B\mathcal{P}_1, & \mathsf{vk}_C &:= \alpha_C\mathcal{P}_2 \\
\mathsf{vk}_\gamma &:= \gamma\mathcal{P}_2, & \mathsf{vk}^1_{\beta\gamma} &:= \beta\gamma\mathcal{P}_1, & \mathsf{vk}^2_{\beta\gamma} &:= \beta\gamma\mathcal{P}_2 \\
\mathsf{vk}_Z &:= Z(\tau)\rho_A\rho_B\mathcal{P}_2, & & & (\mathsf{vk}_{\mathsf{IC},i})_{i=0} &:= \alpha_C\mathcal{P}_2
\end{aligned}$$

and outputs $(\mathsf{pk}, \mathsf{vk})$

Prover $(\vec{z}, \pi) \leftarrow \mathsf{Compute}(\mathsf{pk}, C, \vec{x}, \vec{y})$**:** On input $\vec{x}$ and proving key pk, the prover computes $(\vec{A}, \vec{B}, \vec{C}, Z) = \mathsf{QAPinst}(C)$ as well as a satisfying assignment for the QAP, $\vec{s} := \mathsf{QAPwit}(C, \vec{x}, \vec{y}) \in \mathbb{F}_r^m$.

The prover then samples uniformly $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$. Using these, the prover computes $\vec{h} = (h_0, h_1, \ldots, h_d) \in \mathbb{F}_r^{d+1}$, coefficients of the polynomial

$$H(z) := \frac{A(z)B(z) - C(z)}{Z(z)}$$

where we define $A, B, C \in \mathbb{F}_r[z]$ as follows:

$$\begin{aligned}
A(z) &:= A_0(z) + \textstyle\sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z) \\
B(z) &:= B_0(z) + \textstyle\sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z) \\
C(z) &:= C_0(z) + \textstyle\sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z)
\end{aligned}$$

The prover then defines:

$$\begin{aligned}\tilde{\mathsf{pk}}_A &:= \text{“same as } \mathsf{pk}_A \text{ but with } \mathsf{pk}_{A,i} = 0 \text{ for } i = 0, 1, \ldots, n\\ \tilde{\mathsf{pk}}'_A &:= \text{“same as } \mathsf{pk}'_A \text{ but with } \mathsf{pk}'_{A,i} = 0 \text{ for } i = 0, 1, \ldots, n\end{aligned}$$

Letting $\vec{c} = (1 \circ s \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, the prover computes:

$$\begin{aligned}\pi_A &:= \langle \vec{c}, \tilde{\mathsf{pk}}_A \rangle, \quad \pi'_A := \langle \vec{c}, \tilde{\mathsf{pk}}'_A \rangle, \quad \pi_B := \langle \vec{c}, \mathsf{pk}_B \rangle, \quad \pi'_B := \langle \vec{c}, \mathsf{pk}'_B \rangle\\ \pi_C &:= \langle \vec{c}, \mathsf{pk}_C \rangle, \quad \pi'_C := \langle \vec{c}, \mathsf{pk}'_C \rangle, \quad \pi_K := \langle \vec{c}, \mathsf{pk}_K \rangle, \quad \pi_H := \langle \vec{h}, \mathsf{pk}_H \rangle\end{aligned}$$

and outputs $\pi = (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H)$. Observe that this proof constitutes only eight field elements.

Verifier $\{0, 1\} \leftarrow \mathsf{Verify}(\pi, \mathsf{vk}, \vec{x})$**:** The verifier takes the proof π, the verification key vk, and a public input $\vec{x}$ as its inputs. First, it computes a specialization of the verification key for the input, as follows.

$$\mathsf{vk}_{\vec{x}} := \mathsf{vk}_{\mathsf{IC},0} + \sum_{i=1}^{n} x_i \mathsf{vk}_{\mathsf{IC},\mathsf{i}}$$

Observe that $\mathsf{vk}_{\vec{x}} \in \mathbb{G}_1$.

Next, the verifier checks the validity of the knowledge commitments for A, B, C:

$$e(\pi_A, \mathsf{vk}_A) = e(\pi'_A, \mathcal{P}_2), e(\pi_B, \mathsf{vk}_B) = e(\pi'_B, \mathcal{P}_2), e(\pi_C, \mathsf{vk}_C) = e(\pi'_C, \mathcal{P}_2)$$

The verifier also checks that the same coefficients were used:

$$e(\pi_K, \mathsf{vk}_\gamma) = e(\mathsf{vk}_{\vec{x}} + \pi_A + \pi_C, \mathsf{vk}^2_{\beta\gamma}) \cdot e(\mathsf{vk}^1_{\beta\gamma}, \pi_B).$$

Finally the verifier checks the QAP divisibility condition:

$$e(\mathsf{vk}_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, \mathsf{vk}_Z) \cdot e(\pi_C, \mathcal{P}_2 x).$$

The verifier accepts (i.e., outputs 1) if and only if all of these checks pass. Otherwise, it rejects (i.e., outputs 0).

The above protocol relies on (for polynomial $q(|C|)$): (i.) the q-power Diffie-Hellman assumption; (ii.) the q-power knowledge-of-exponent assumption; and (iii.) the q-strong Diffie-Hellman assumption citeBB04, gennaro2004multi, groth2010short. It is worth observing that the foregoing protocol is *succinct* in the sense that proofs are very short and verification of those proofs is very fast—implementations of the above protocol, which we use in Chapter 6, can achieve 288-byte proofs at 128 bits of security (or 230-byte proofs at 80 bits of security), with verifications requiring only approximately 5ms on modern hardware. Observe as well that the work of the verifier does not depend on $|C|$, and therefore that verification effort does not depend on the complexity of the input computation. KeyGen and Compute, however, are heavyweight steps that do depend on $|C|$

It is also worth observing that the foregoing protocol requires the KeyGen algorithm to be executed by a trusted party—a party who knows the random coins used by this algorithm can straightforwardly forge proofs that pass verification but do not represent possession of the underlying knowledge. That is, such a party can break the soundness of the proof system. Therefore, it is important that the KeyGen process be trustworthy. It can either be executed by a trusted party, which need not be present after the KeyGen step is complete and the keys published, or it can be computed in a distributed fashion using the distributed, multiparty techniques of Ben-Sasson et al. [40]. We remark that the distributed key generation approach significantly improves the deployability of these tools, especially for accountability systems, which

must engender trust in their correct operation by demonstrating that parties that misbehave will be detectable.

3.3.3 Pseudorandomness

Pseudorandomness aims to generate long sequences of bits, using a deterministic process and short sequences of randomly selected bits, which are *computationally indistinguishable* from long sequences of truly random bits. Specifically, we define below *pseudorandom generators* (PRGs) which achieve this goal. Our main protocol, presented in Chapter 5, uses a PRG to expand a small, verifiable random seed into a long string from which any random coins needed by some computation can be drawn. This allows random choices, which need to differ over many executions of a program, to be replaced by short, recordable strings that can be generated in a trustworthy way. We also describe the related idea of *pseudorandom functions* (PRFs), which are efficiently representable, efficiently computable functions which are computationally indistinguishable from functions selected at random from the space of functions with the same domain and range. We further describe the construction of *verifiable random functions* (VRFs), which produce a third-party verifiable proof of their correct execution. We use a VRF in our protocol in Chapter 5, instantiating it with an efficient VRF due to Dodis and Yampolskiy [127] in our implementation. We describe the construction of this concrete VRF as well. We end the section by summarizing the literature on verifiable fair sampling, which we depend on for our work in later chapters but do not explicitly use in our protocols.

Computational Indistinguishability

Before beginning our discussion of primitives for pseudorandomness, however, it is useful to define formally the notion of computational indistinguishability, which captures the idea that no efficiently computable process can distinguish between two

distributions. Once we establish that two distributions, such as one that samples uniformly at random and one that samples pseudorandomly, are computationally indistinguishable, we can consider objects which sample from one distribution as equivalent to objects that sample from the other. Indeed, if we had a process in which we could not replace one an object that samples randomly by an object that samples pseudorandomly, this process would yield an efficient distinguisher for the pseudorandomness of the replacement distribution, contradicting our initial assumption. This is the basis for much of modern cryptography. We follow Goldreich [168, Ch. 3] in giving definitions for abstract primitives and computational indistinguishability.

We begin by defining the notion of a *probability ensemble* [168, Defn. 3.2.1].

Definition 4. *Probability Ensemble Let I be a countable index set. An* **ensemble indexed by** *I is a set of random variables indexed by I, $X = \{X_i\}$ where $i \in I$.*

We require this notion to define computational indistinguishability [168, Defn. 3.2.2,#1]. Specifically, we use $I = \mathbb{N}$ below, although it is also possible to define probability ensembles indexed by (e.g.) binary strings of up to a specific length. These definitions can be unified by associating natural numbers with their unary representations (i.e., associating $\mathbb{N}$ and $\{1^n\}_{n\in\mathbb{N}}$.

Definition 5. *Polynomial-Time Indistinguishability Two ensembles $X = \{X_i\}_{i\in\mathbb{N}}$ and $Y = \{Y_j\}_{j\in\mathbb{N}}$ are* **indistinguishable in polynomial time** *if for every probabilistic polynomial time algorithm D, every positive polynomial p, and all sufficiently large n, we have*

$$\Pr D(X_n) = 1 - \Pr D(Y_n) = 1 < \frac{1}{p(n)}$$

The probabilities in the above definition are taken over the random variables X_i, Y_i and the internal coin tosses of the algorithm D. We remark that this definition is related to but coarser than the concept of *statistical closeness*. In particular, given a domain for a random variable, there exists a distribution which is statistically far from

the uniform distribution over that domain, but which is nonetheless indistinguishable from it.

Pseudorandom Generators (PRGs)

Our definitions tell us that we can replace randomly sampled probability ensembles by pseudorandom ensembles with at most a negligible cost to the performance of an efficient application that reads the ensemble. However, such replacement is only useful if we can sample pseudorandom ensembles more efficiently than we can sample truly at random. This gain is achieved by pseudorandom generators, which we define in this section, again following Goldreich [168, Defn. 3.3.1].

Definition 6. *Pseudorandom Generator (PRG) A* **pseudorandom generator (PRG)** *is a deterministic polynomial time algorithm G satisfying:*

(i.) Expansion*: There exists a function $l : \mathbb{N} \longrightarrow \mathbb{N}$ such that $l(n) > n$ for all $n \in \mathbb{N}$ and $|G(s)| = l(|s|)$ for all $s \in \{0,1\}^*$.*

(ii.) Pseudorandomness*: The ensemble $G = \{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom. That is, there exists a uniform ensemble $U = \{U_n\}_{n\in\mathbb{N}}$ such that G and U are indistinguishable.*

The function l is called the expansion factor *of G.*

Pseudorandom generators exist under the widely held assumption that one-way functions exist. The input to a pseudorandom generator is usually referred to as a *seed*.

We realize PRGs in our work in Chapter 6 using cryptographic sponge functions [190], specifically the function Keccak [44], now the SHA-3 standard. Sponge functions work by *absorbing* input in blocks to build an internal *state*. Once the sate is constructed, the sponge can be *squeezed* to emit blocks of pseudorandom bits. As

they are somewhat intricate, we refer the interested reader to the presentations of Keccak and sponge functions generally in the reference and standards documents.

Pseudorandom Functions (PRFs)

Pseudorandom functions (PRFs) are efficiently computable functions that are indistinguishable from an ensemble of functions that selects uniformly from the set of functions with the same domain and range. In this way, a pseudorandom function cannot be efficiently distinguished by an adversary from a randomly sampled function. This makes the functionality of a PRF similar to that of a PRG, and in fact it is straightforward to construct one from the other.[11] However, PRFs give us a way to generate poly(n) pseudorandom values assigned to n-bit strings, and retrieve those bits dynamically, while storing only n bits over the life of an application. We use a variant of PRFs described below, verifiable random functions, to achieve exactly this: in Chapter 5 we wish to generate a pseudorandom value for each subject of an automated decision process based on the characteristics of that subject, the process, and a truly random value supplied from a trusted process. This gives us a reproducible source of cryptographic-quality per-subject seeds for use in a PRG.

We use a definition of pseudorandom function families due to Lynn:[12]

Definition 7. *Pseudorandom Function Family A function* $f : \{0,1\}^n \times \{0,1\}^k \longrightarrow \{0,1\}^m$ *is a* **pseudorandom function family**, *parametrized over its first input, if given a key* $K \in \{0,1\}^k$ *and an input* $X \in \{0,1\}^n$, *there is an efficient algorithm to compute* $f_k(x) = f(x,k)$ *and for any probabilistic polynomial time oracle machine* $A^{(\cdot)}$, *we have:*

$$|\mathsf{Pr}_{K \leftarrow \{0,1\}^k}[A^{f_K} = 1] - \mathsf{Pr}_{f \in \mathcal{F}}[A^f = 1]| \leq \epsilon$$

[11] Goldwasser, Goldreich, and Micali [170] show how to efficiently construct PRFs from a PRG. The opposite construction is trivial: just evaluate a PRF at successive points $(F_k(1), F_k(2), \ldots)$ to obtain the desired amount of output.

[12] In addition to his excellent description of the details of implementing pairing-based cryptography [256], Lynn maintains an extensive set of notes on general topics in cryptography at `https://crypto.stanford.edu/pbc/notes/`

where $\mathcal{F} = \{f : \{0,1\}^n \longrightarrow \{0,1\}^m\}$.

PRFs can be used to construct simple symmetric encryption schemes and secure message authentication codes, however we do not need them for this—only to construct verifiable random functions, described below.

We remark that even an average-case random function is very inefficient, requiring nearly all of its input-output relation to be specified explicitly in its shortest representation. The construction of efficiently computable pseudorandom functions, even under the assumption of one-way functions, is one of the major achievements of modern cryptography and computer science generally.

Verifiable Random Functions (VRFs)

We can extend the notion of a PRF to define verifiable random functions (VRFs). Intuitively, VRFs behave like PRFs, but also output a proof of their outputs' correctness. More formally, we follow the definition given by Dodis and Yampolskiy [127, Def. 1]:

Definition 8. *Verifiable Random Function Family A family of functions* $F_{(\cdot)}(\cdot) : \{0,1\}^n \longrightarrow \{0,1\}^n$ *is a family of VRFs if there exists a probabilistic polynomial time algorithms* Gen *and deterministic algorithms* Prove *and* Verify *such that* $\mathsf{Gen}(1^\kappa)$ *outputs a pair of keys* $(\mathsf{sk}, \mathsf{vk})$*;* $\mathsf{Prove}_{\mathsf{sk}}(x)$ *computes* $(F_{\mathsf{sk}}(x), \pi_{\mathsf{sk}}(x))$*, where* $F_{\mathsf{sk}}(x)$ *is a pseudorandom function and* $\pi_{\mathsf{sk}}(x)$ *is a proof of correctness in the sense that if* $(y, \pi) = \mathsf{Prove}(\mathsf{sk}, x)$*, then* $\mathsf{Verify}_{\mathsf{vk}}(x, y, \pi) = 1$*. Also, no values* $(\mathsf{vk}, x, y_1, y_2, \pi_1, \pi_2)$ *may satisfy* $\mathsf{Verify}_{\mathsf{vk}}(x, y_1, \pi_1) = \mathsf{Verify}_{\mathsf{vk}}(x, y_2, \pi_2)$.

Under this definition, the proof in a VRF serves to convince the verifier that the announced output of the VRF corresponds to the specified input and not some other input.

Dodis and Yampolskiy construct a VRF[13] under the q-Diffie-Hellman Inversion assumption, which states that it is hard, given $(g, g^x, g^{x^2}, \ldots, g^{x^q})$ as input, to compute $g^{1/x}$. Their construction works as follows, given public parameters p a prime and $g \in \mathbb{Z}_p$.

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k)$**:** The key generation process chooses a secret $s \xleftarrow{\mathbb{Z}^*_p}$ and sets $\mathsf{sk} = s, \mathsf{vk} = g^s$.

$(y, \pi_{\mathsf{sk}}(x) \leftarrow \mathsf{Prove}_{\mathsf{sk}}(x)$**:** The function is evaluated by setting the output $y = g^{1/x+\mathsf{sk}}$. The proof in this particular construction is embedded in the output value, so no separate proof is necessary.

$\{0, 1\} \leftarrow \mathsf{Verify}(x, y, \pi)$**:** The verifier checks that $e(g^x \cdot \mathsf{pk}, y) = e(g, g)$ and outputs 1 if so and 0 otherwise. Indeed, if the function was computed correctly:

$$e(g^x \cdot \mathsf{pk}, y) = e(g^x g^s, g^{1/(x+s)}) = e(g, g)$$

This scheme is proposed as a signature scheme by Boneh and Boyen [59]. Dodis and Yampolskiy prove that it is verifiable and satisfies a technical "unpredictability" criterion, making it a verifiable unpredictable function (VUF). This is sufficient for our pseudorandomness and verification needs in the protocol in Chapter 5.

3.3.4 Fair Randomness

In order to take advantage of the tools for pseudorandomness introduced in Section 3.3.3, it is still necessary to sample a small number of bits truly at random or at least in a way that is beyond the control and predictive power of parties to a protocol.

[13]Technically, the construction we describe here is only a verifiable *unpredictable* function, not a verifiable random function. However, with some care in its application, it suffices for the purposes for which we require it in Chapters 5 and 6. Specifically, we must be careful not to interpret all of the output of the VUF as pseudorandom, but must discard some of it using standard techniques, which we describe in Chapter 6.

In this section, we consider the large existing literature on techniques for generating verifiable *fair randomness*, which achieve this goal. We observe that there are a few key approaches to this problem which exist in the literature: *public ceremonies*, such as lotteries, to draw lots and assign scarce resources have been used since ancient times; one can simply trust an *entropy authority* to produce valid randomness—when such an authority publishes its randomness widely, it is often referred to as a *random beacon*; finally, modern cryptography has produced a wealth of multiparty protocols, including the distributed key generation protocols mentioned in Section 3.2.3, which generate strings outside of the control of their participants and which have the important property that, so long as at least one participant behaves honestly, the entire protocol execution remains trustworthy.

Public Randomness Ceremonies

Lotteries have a rich and ancient history: the use of lotteries to raise funds for state projects (e.g., many authors describe the use of a keno-like game by the Han Dynasty to raise funds for its Great Wall project) and to assign scarce resources; Price surveys knowledge of these and other early Asian uses of games of chance in the context of a social analysis of gaming and gambling in traditional Asian cultures more generally [302]. They were also used in early Celtic cultures, mentioned by Homer in the Iliad, and run as an amusement during dinner parties in Ancient Rome. Later, lotteries in medieval times were used to raise money for public works projects including fortifications. In early modern times and into modern times, lotteries of various sorts were employed by most large European states and now exist worldwide. Ezell gives a more extensive history [140].

Lotteries as public ceremonies for generating a random outcome can be as predictable as placing a set of tickets or tokens representing outcomes in a bin or hat, or depend on more complicated mechanisms such as throwing wooden pegs onto a board.

Modern instantiations often use a machine that tumbles small plastic balls numbered to encode outcomes and which selects one or more balls without replacement to determine the lottery outcome. However, even the use of mechanized tumbling has failed to produce results indistinguishable from random in several notable cases. One such case was the failure of the lottery for the United States Selective Service System in 1969 [318, 340], which produced measurably biased output such that participants with birthdays in November and December were more likely to receive low numbers representing early induction into military service. Another case is the failure of the fiscal year 2012 Diversity Visa Lottery operated by the U.S. State Department, which we consider in more detail in Chapter 6.

Beyond lotteries, public ceremonies are used to generate and share cryptographic keys in a verifiable manner, as analyzed in detail by Ellison [134]. Considered more abstractly, ceremonies for the fair exchange of high-value objects such as treaty signatures have an ancient history as well.

Random Beacons and Trusted Authorities

The concept of relying on a broadcast source of randomness to protect the integrity of protocols requiring a random input is due to Rabin [305], who gives one of the earliest protocols using this technique.

An alternative approach is suggested by Corrigan-Gibbs et al. [103], who suggest that trusted authorities can supply entropy for mixing in key generation and then can receive a zero-knowledge proof that this entropy was incorporated into key generation. In this way, the authority can later certify that generated keys were drawn from a large space.

Backes et al. present CSAR, a system for cryptographically strong accountable randomness authorities [21]. CSAR introduces authorities who provide randomness along with a protocol to hold those authorities accountable for a set of security in-

variants designed to make interaction with the authority indistinguishable from interaction with a trusted beacon.

Multiparty Fair Randomness: Coin flipping over the phone

The largest suite of technical tools for building verifiable fair randomness comes in the form of multi-party protocols for mixing purported randomness from many participants. The properties of pseudorandom functions and pseudorandom generators make it easy to take a variety of randomness sources and "mix" them to create pseudorandom values which are not predictable to any participant and which remain pseudorandom as long as at least one participant submits an actual random value.

The earliest such protocol is Blum's famous technique for "coin flipping over the phone" [54]. In the protocol, each player commits to a random string and passes the other player a commitment to that string. The players then open their commitments and combine the randomness (for example, if the committed value is a single bit, the bits of the two players can simply be added in $\mathbb{Z}_2$. Longer strings can be combined bit-wise.

More complicated versions of the above basic technique can admit many players and yield the generation of structured random objects, as in the distributed key generation protocols described in Section 3.2.3. We remark in particular on a protocol for securely sampling the large, circuit-specific keys needed in the trusted setup phase of deploying a zk-SNARK system, due to Ben-Sasson et al. [40], which makes our zk-SNARK realization of the protocol presented in Chapter 5 much more realistic and deployable.

Chapter 4

Accountable Warrant Execution

In this chapter, we consider how to achieve accountability for an example application with a well-defined policy that can be expressed completely in advance: the problem of restricting compelled access to data by law enforcement. This protocol is joint work with David Wu, Joe Zimmerman, Valeria Nikolaenko, Dan Boneh, and Edward W. Felten and has been released online in a preliminary form [232]. Because in this application it is clear that law enforcement should only access data under a valid order, we can design a system where this property is maintained as an invariant. One such design is the focus of this chapter; it is meant as an exemplar of a wide literature of such "correct by construction" systems (see Chapter 2, Section 2.1 for a high-level overview of these approaches).

4.1 Problem and Setting

Law enforcement increasingly demands access to data held by the providers of online communications services,[1] sometimes demanding access to a wide swath of commu-

[1]See, for example, the graphic depicting the relative volume of data production between several popular services in "Thanks, Snowden! Now all the major tech companies reveal how often they give data to government." Kashmir Hill, Forbes. `http://www.forbes.com/sites/kashmirhill/2013/11/14/silicon-valley-data-handover-infographic/`

nications metadata such as the details of when and between whom communications take place.[2] In many cases, such compelled access requires a legal grant of authority such as a *warrant*.[3] However, despite efforts by service providers to be transparent about the requests they receive and respond to,[4] there is still no way for outsiders to verify that law enforcement can justify the origin of any data it collects. In particular, even the overseers of such investigation activities cannot verify that the information produced responsive to some legal authority or order actually matches the scope of that authority, which has led to abuses of access in the past.[5]

[2]For example, The U.S. National Security Agency (N.S.A.) was disclosed in 2013 to be collecting all "call detail record information" from a major phone carrier on "an ongoing daily basis". See Greenwald, Glen "NSA collecting phone records of millions of Verizon customers", The Guardian, 6 June 2013. `http://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order`

[3]There are many sources of authority under which law enforcement may compel access to data, of which a warrant is only one kind. For example, many kinds of data are available under a panoply of administrative subpoenas depending on the jurisdiction and the type of data requested. For an overview, see Jonathan Mayer's course on Surveillance Law, `https://www.coursera.org/course/surveillance` or the works of Kerr [226–228].

[4]*cf.* the "transparency reports" of several popular online service providers:

(i.) Google: `https://google.com/transparencyreport`;

(ii.) Twitter: `https://transparency.twitter.com/`;

(iii.) Yahoo!: `https://transparency.yahoo.com/`;

(iv.) Facebook: `govtrequests.facebook.com`;

(v.) Microsoft: `https://www.microsoft.com/about/corporatecitizenship/en-us/reporting/transparency` and `https://www.microsoft.com/about/corporatecitizenship/en-us/reporting/fisa`;

(vi.) Apple: `https://www.apple.com/privacy/transparency-reports/` and `https://www.apple.com/privacy/government-information-requests/`;

(vii.) Dropbox: `https://www.dropbox.com/transparency/`

(viii.) CloudFlare: `https://www.cloudflare.com/transparency`

[5]In particular, see disclosures of formal reprimanding of the U.S. National Security Agency by its overseer for purposes of data access, the Foreign Intelligence Surveillance Court,

(i.) Charlie Savage and Scott Shane, "Secret Court Rebuked N.S.A. on Surveillance", The New York Times, 21 August 2013. `http://www.nytimes.com/2013/08/22/us/2011-ruling-found-an-nsa-program-unconstitutional.html`

(ii.) Scott Shane, "Court Upbraided N.S.A. on Its Use of Call-Log Data", The New York Times, 10 September 2013. `http://www.nytimes.com/2013/09/11/us/court-upbraided-nsa-on-its-use-of-call-log-data.html`

Further, such compelled access is indistinguishable technically from an *insider attack*, in which a malicious insider gathers the same records and provides them to an adversary either under duress or in response to a bribe. Systems which admit access for insiders to respond to legal orders are naturally susceptible to these sorts of attacks, and securing against insider abuses while maintaining flexibility in access control policy choices remains an important open problem. Because observers cannot tell the difference between compelled access and an insider attack, the accountability of such systems suffers.

In this chapter, we propose to address these concerns with *accountable warrant execution*, a cryptographic assurance that, subject to certain assumptions about trust, if an *investigator* requests permission from a *court* to compel the production of *records* from a *data source* that pertain to a specified *identifier*, and the court approves that request with a valid *order*, then the records seen by the investigator in response to its request actually correspond to the scope of the granted order. In particular, the records in our protocol are held encrypted, and the power to enable the investigator to decrypt and review them rests with a set of independently trusted *decryption authorities*. Every decryption by the decryption authorities in response to a valid order by the court results in an *audit record*—a committed, encrypted version of the order—which is reviewed by a designated *auditor*.

Our protocol provides strong guarantees of *accountability*, in the sense that an after-the-fact review of the audit records can always be used both to justify any plaintext records in the possession of the investigator and to effectively blame a specific misbehaving party if the protocol is ever violated. Our goal in ensuring accountability is to facilitate the existing social, political, and legal processes designed to constrain and oversee the execution of legal orders by the investigator.

Accountability in this sense is about ensuring that the behaviors of the parties to the protocol are well explained and well justified so that an oversight process can

detect misbehavior and parties to the protocol will pay a cost for deviating from the rules. Hence, the parties will wish to follow the protocol faithfully. It is separately necessary for the protocol to ensure other important *correctness* properties, such as the information flow restriction that the investigator only see records which have been decrypted in response to a valid order. Clearly, if a data source wants to share plaintext directly with the investigator, we cannot detect or stop this—no protocol can prevent the sharing of information outside that protocol's formal scope.

Under appropriate assumptions about trust in the parties, described in our security model in Section 4.2, the protocol satisfies several critical properties:

- The protocol grants the investigator access to exactly the records authorized by the court via an order.
- Every decryption generates an audit record.
- Every decrypted record held by the investigator can be correlated to an audit record by the auditor.
- An unprivileged viewer of the audit record learns that all access was *correct* in the sense that it was authorized by the court, without learning precisely which records were accessed or under what justification. That is, the existence and scope (in the sense of the number of record identifiers) of any compelled access is *publicly verifiable*, even if accesses to specific records is only verifiable by the court or auditor. The number of orders is also revealed, such that (subject to assumptions about the granularity of orders) the public can know how many records the investigator is reviewing.
- A privileged viewer of the audit record (i.e., a designated oversight entity such as a court or legislature) can learn additional information: precisely which records were accessed and the justification for that access. Such an overseer also learns

whether the audit record is *valid*, or consistent with the requests made by the investigator.

Our protocol provides the tools necessary to solve this problem while enabling effective oversight: the investigator can request access to specific, particularized, identified records from a court authorized to determine the validity of that access; the fact of these requests is always logged for future audit; *and* a disinterested third party can learn that the investigator's actions are properly logged for review and oversight, without learning the specific operational details of each request. Indeed, in our protocol, the details of the investigator's requests are only learned by the investigator itself and the court, while the data sources and decryption authorities learn only the existence of valid orders for which access was granted.

Our protocol realizes a new cryptographic notion, which we call *auditable oblivious transfer* (aOT), which is of independent interest. Our aOT derives from the *adaptive oblivious transfer* of Green and Hohenberger [183], introduced in Chapter 3, Section 3.2.2. aOT allows a *sender* with messages $m_1, \ldots, m_k$ to transmit to a *receiver* with an index $1 \leq \ell \leq k$ the message m_ℓ such that the sender learns no information about ℓ except an auditing record which is a commitment to ℓ and an encryption under the key of a designated auditor of the randomness used to generate this commitment. We extend the notion of aOT to auditable threshold oblivious transfer, in a setting with multiple senders, as we describe in Section 4.3.

4.1.1 Applications

Although we use terms like "investigator", "court", and "order" to describe our protocol, we observe that the protocol has many practical applications. We summarize a few here.

Law enforcement or intelligence agency access to communications metadata or business records An intelligence analyst (the investigator) requests access to business records (e.g., phone call metadata) held by a carrier (the data source) for a targeted person (named in the identifier) from a special intelligence court (the court). The court will determine whether that access is justified under the law according to the relevant standard, such as Reasonable Articulable Suspicion, and allow access if so, sending this justification to an oversight body (the auditor).

Cloud provider access to customer data An employee at a cloud provider (the data source) requests access to customer data for a specific account (the identifier) from the company's general counsel (the court). In this example, oblivious decryption is likely unnecessary: if some trustworthy portion of the cloud provider's organization (such as an internal oversight board) is acting as a decryption authority, it might be possible or even desirable for that entity to learn which account was decrypted.

Authorized Access to Sensitive Audit Records Many situations exist where audit records are generated as a matter of course, but are only intended to be reviewed in exceptional circumstances. We imagine a system (the data source) which generates privacy-sensitive audit records (e.g., surveillance videos) indexed by some identifier (e.g., time, the camera location) and those records are only reviewed when some authority figure (the court, e.g., a manager) certifies that a set of exceptional criteria have been met (the order, e.g., the cameras have recorded a potential crime). Again, oblivious decryption may not be a requirement in this scenario—a technician instructed to review audit records might know or have a reasonable guess as to which records correspond to exceptional circumstances (e.g., the location of a video surveillance camera which recorded a crime and the approximate time of the relevant footage).

4.1.2 Our Results

This chapter introduces a protocol for accountable compelled access to data, which achieves a strong notion of accountability that reflects our high-level goal of facilitating oversight by people of a technical process and also a strong formal notion of accountability for cryptographic protocols. Further, we implement this protocol and show that it is practical for realistic application scenarios, such as the oversight of law enforcement or intelligence use of telephony metadata records for a large country. In particular, we show that it is possible to encrypt half a billion records using our implementation in just under three hours using approximately one standard rack of modest hardware. Our protocol is practical both for normal operation (i.e., encryption) and supports the execution of court orders (i.e., decryption) in only a few seconds, even with a large number of decryption authorities.

Structure of the Chapter In Section 4.2, we describe our security model and the specific properties achieved by our protocol. We also describe a simple protocol which does not fully meet our requirements. Using the cryptographic primitives introduced in Chapter 3, Section 3.2, Section 4.3 introduces a new notion that provides a useful formalization of what our protocol achieves: auditable oblivious transfer. Section 4.4 describes our full protocol. Section 4.5 describes our implementation of the protocol, which we benchmark in Section 4.6. We summarize related work specific to this chapter in Section 4.7 and conclude in Section 4.8 by discussing some possible extensions to our protocol as well as practical difficulties in selecting trustworthy decryption authorities for law enforcement applications.

4.2 Security Model

We give a brief overview of the model for our analysis: the parties to our protocol, our trust assumptions, the security properties we desire, and a discussion of the meaning and usefulness of accountability specific to the approach described in this chapter.

4.2.1 Setting

Our protocol applies in the context of an investigator serving legal orders for compelled access to specific labeled data records, $x_{i,j,t}$. We assume that records are indexed by a per-record *tag*, in our case a triple (i, j, t). A record's tag contains: an index i identifying the data source where it originated; an index j identifying the user to which it pertains; and a value t which represents the time interval in which the record was created or collected (this scheme allows flexibility in clustering records by time interval—t could be a date or any other time period, giving orders the granularity of "all records produced in period t"; but t could also simply be a sequence number which is different for each new record). Our general approach will be to encrypt each record under a key specific to its tag and then to let access to these keys be a proxy for access to the underlying plaintext records, controlling and auditing that access as necessary to meet our security goals.

Parties Our protocol has five types of parties (see Figure 4.2), which we model as computationally bounded communicating processes in the standard cryptographic model:

S_i: **Data Sources** such as telecommunications carriers or cloud service providers, who generate, collect, and label records $x_{i,j,t}$. The role of each data source in our protocol is to encrypt each record it collects under a key specific to the tag of that record.

I: the **Investigator** seeks legally authorized compelled access to the records produced by the S_i.

C: the **Court** is the designated authority with the power to approve or deny the investigator's requests and the arbiter of access to the data records.

D_k: the **Decryption Authorities** hold secret key material that allows the decryption of authorized records. Our protocol requires t of N decryption authorities to participate in the threshold decryption of a record. Each authority participates only when presented with a valid order for decryption. Decryption authorities may or may not be authorized to know which specific tag (i, j, t) is being decrypted; we assume unless otherwise stated that each D_k must be blind to (i, j, t). Decryption authorities should be independently trusted, such that the compromise of secret key material from one does not affect the integrity of the others.

$\mathcal{A}$: the **Auditor** maintains a log of the protocol execution and certifies the presence of entries in this log to other parties, who do not allow the protocol to make progress without such certification. In particular, the auditor retains audit records generated as the protocol progresses. We describe the auditor as a party to the protocol as this more closely matches our desired setting, where audit records are reviewed by a privileged oversight body, but it would be sufficient for the auditor's online role to be implemented as an append-only log visible to all other parties.

Trust assumptions We assume the parties described above represent independent bases of trust and will not be mutually compromised or collude. Further, we assume that all parties will follow the protocol honestly if they believe misbehavior will be detected. We also assume that the parties can gather in a trusted environment to generate and exchange keys (e.g., for signature schemes and channel authentication) or can rely on a trusted authority or PKI to generate and exchange such keys on their behalf. Finally, we must make the (admittedly strong) assumption that all

interactions between the parties are as defined by the protocol (that is, we assume that the parties do not take actions "off the books").

The strength of these assumptions is tempered by the value of accountability, described in Section 4.2.2. For example, the fact that the investigator must be able to justify to an oversight body the origin of any records in its possession and the assumption that the investigator does not wish to be caught misbehaving imply together that the investigator will always request access to records using the protocol.

Communication Model In this chapter, we assume that all parties communicate over confidential and mutually authenticated channels. Such channels can be efficiently realized using, e.g., TLS [122].

4.2.2 Security Goals

We briefly describe the security invariants a protocol in our setting must provide, which are all achieved by the construction presented in Section 4.4.

Record Secrecy At a basic level, we require that the data records be kept secret from all parties and from any third party who observes either the protocol itself or the audit log generated by the protocol. The only exceptions to this rule are that the records may be seen by the data source S_i which originates them (that is S_i can access records with tags (i, j, t) for all j and t) and the Investigator I can access precisely the set of records for which it has requested and received access via the court C under a valid, recorded order.

Accountability At an intuitive level, we wish to guarantee the invariant that the investigator gets access to all and only the records authorized by the court. We also require that the investigator be able to *justify* any records it holds by identifying a corresponding audit record demonstrating that it was obtained through proper

channels. Finally, we require that if the record secrecy invariant is violated or that if some party fails to fulfill its duties under the protocol, that the protocol transcript up to the point of violation provides enough information to an unprivileged third-party observer that the observer can blame a specific party for misbehavior.[6]

These intuitive notions of *accountability* are similar to the formal definition introduced by Küsters, Truderung, and Vogt [235] in that inspection of the transcript of our protocol (or, in our case, a subset, namely the logged *audit record*) will allow any *observer* to *blame* a specific misbehaving party in the event any party misbehaves.

We define accountability for the purpose of this dissertation in a different, more high-level way: namely that the social, political, and legal processes designed to constrain and oversee the execution of court orders can operate effectively. This is similar to the Küsters et al. notion of *goal-based accountability*, but requires the additional invariants described above, namely that the investigator be able to justify any record to which it has access by exhibiting an order authorizing access to that record. Both notions of accountability are stronger than the commonly considered properties of *auditability* or *verifiability*. Chapter 2, Section 2.1.8 describes this distinction further.

Value of Accountability By defining accountability as a property not just of our protocol, but our protocol in the context of human-level processes, we create a robust link between the formal guarantees provided by the protocol and the question of why the parties are deterred from violating the protocol detectably. Indeed, since we cannot prevent the parties from sharing extra plaintext outside the protocol, we must rely on the parties' incentives to avoid detectably violating the rules and hence on the parties' accountability to subsequent oversight.

[6]While these notions do not capture certain types of misbehavior such as *parallel construction* (in which the investigator learns the contents of a specific record before requesting it and subsequently requests that record in order to satisfy the accountability requirements), such misbehavior is inherently beyond the scope of the sort of access control possible with any protocol of the type we introduce.

While we do not take a position on the correctness of any policy that allows compelled access to private data by law enforcement, we emphasize that there are risks inherent to the collection of private data by data sources. Our protocol aims to minimize the additional risks of abuse or mishap due to the need to comply with compelled access orders in the case where data are already being collected by the data source. This chapter, and indeed this dissertation, does not argue in favor of mandatory collection or retention of data, nor do we express any opinion on when law enforcement or intelligence agencies should be able to demand access to data. In particular, we believe our protocol is better suited to deployment scenarios in which data are already collected or are collected as part of the natural function of another process, such as in the telephony metadata example described in Section 4.1.

Robustness to partial compromise Our protocol must continue to provide strong accountability and record secrecy guarantees in the event that one or more parties is compromised. Specifically, the protocol must resist compromise or collusion of any subset of the court, the auditor, and any subset of the decryption authorities insufficient to decrypt records.

We guarantee these properties in an *offline* sense: misbehavior can always be detected, but might only be detected after the fact. In Section 4.4, we discuss how to construct an extension of our protocol which achieves *online* accountability (i.e., if any party misbehaves, the other parties will notice immediately and can abort the protocol). Regardless, misbehavior should be visible to a non-participant in the protocol who reviews the audit record.

While our model allows some level of compromise or collusion, we stress that because the goal of our protocol is accountability, such issues must be eventually detectable, but perhaps only by review of the behaviors of specific parties by their respective oversight bodies.

Audit independence Our protocol must satisfy a strong notion of *third-party auditability*. That is, any non-participating third party who observes the audit record produced by the protocol should, subject to the trust assumptions outlined above, be able to determine that the record secrecy and accountability properties described above hold. Further, a privileged third party who has oversight authority should be able to learn precisely which records were requested, which requests were authorized, and which record keys were produced.

4.2.3 A simple, insufficient approach

In this section, we describe a temptingly simple approach to our problem. However, it achieves only some of the above properties. This protocol is depicted in Figure 4.1.

Protocol 1 ((Insufficient)).

Setup. The court and the auditor each generate a public/private key pair (($\mathsf{pk}_*, \mathsf{sk}_*$) and ($\mathsf{pk}_\mathcal{A}, \mathsf{sk}_\mathcal{A}$), respectively) for a public-key encryption system, with the auditor's secret key $\mathsf{sk}_\mathcal{A}$ known as the *escrow key*. The auditor also generates a signing/verification key pair. We assume the public and verification keys are known to all parties, but that the secret and signing keys are held only by the generating party.

Normal Operation. Each data source S_i encrypts its data records $x_{i,j,t}$ under a fresh key pair ($\mathsf{pk}_{i,j,t}, \mathsf{sk}_{i,j,t}$) for a labeled public-key encryption scheme [336], using the record's tag as the label, and sends (or makes available) the resulting ciphertext $\mathsf{ct}_{i,j,t}$ to the investigator along with an encryption $\mathsf{ct}_{\mathsf{pk}_{i,j,t}}$ of the record key under the court's master key pk_*.

Investigator Query. When the investigator wishes to decrypt a particular record ciphertext $\mathsf{ct}_{i,j,t}$, it sends a request consisting of the requested record tag (i, j, t), and the associated record key ciphertext $\mathsf{ct}_{\mathsf{pk}_{i,j,t}}$ to the court asking it to decrypt that

ciphertext. The court either approves or denies this request. In either case, the court encrypts the request as well as its decision under $\mathsf{pk}_{\mathcal{A}}$ and sends it to the auditor, which replies with an acknowledgment that includes a signature σ_{L} on the court's log message. If the court has denied the investigator's request, the protocol ends here.

If, however, the court has approved the investigator's request, the court verifies that the signature σ_{L} corresponds to the court's audit message and hence to the investigator's original request for the record key for tag (i, j, t). If the signature verifies, the court will use its master secret key sk_* to determine $\mathsf{pk}_{i,j,t}$ by decrypting the provided record key ciphertext using the provided tag as the label.

Protocol 1 is very simple: the court plays both the role of the Court and the role of a Decryption Authority as described in Section 4.2.1. Because of this, the protocol is not robust to compromise of or misbehavior by the court, either of which will allow arbitrary decryption of records without any sort of enforcement that the relevant audit record will be produced, violating the record secrecy and accountability requirements stated in Section 4.2.2. We can improve the security posture of the court in various ways, such as by equipping it with a hardware security module initialized with the secret key sk_* to minimize the risk of exfiltration. However, such enhancements do not fundamentally alter the fact that this protocol fails to meet our robustness against partial compromise requirement and in fact fails to give accountability in the case of a misbehaving or compromised court.

This protocol also does not provide the audit independence property described in Section 4.2.2. This property could be achieved by extending the protocol so that the court also submits to the auditor a separate non-interactive zero knowledge proof that the key it decrypted, $\mathsf{pk}_{i,j,t}$ and the tag in the court's audit message (i, j, t) match the tag in the investigator's request. To make such a proof efficient, it is necessary to chose the public-key encryption scheme in the protocol such that it admits an efficient proof of this knowledge.

Additionally, Protocol 1 is not very efficient, requiring the data source or the investigator to store one record key for each encrypted record. The record tag must also be stored with each ciphertext in the clear, which may render the privacy benefits of encrypting the record itself useless.

The careful reader will notice that the use of record-specific keys in Protocol 1 is not strictly necessary, since the data sources could just encrypt the records themselves under pk_*. We present the protocol as above for two reasons: first, the use of record-specific keys hews closer to our high-level model of allowing per-record keys to represent capabilities for access to a particular record. Second, using this hierarchical key architecture prevents the court itself from viewing the records, allowing us to satisfy a more stringent record secrecy policy than in the simplest case.[7]

Below, in Section 4.2.4, we describe a more robust protocol, which is described formally in Section 4.4. Naturally, this robustness comes with some complexity; the primitives necessary to build it were presented in Chapter 3. Our new primitive, auditable oblivious transfer, is presented below in Section 4.3.

4.2.4 A complete approach

The straw man protocol in Section 4.2.3 was insufficient because: (i.) The court is a trusted single point of failure and this fails our robustness requirement; (ii.) the protocol cannot be audited by an untrusted observer, which fails our audit independence requirement; and (iii.) the number of keys in the protocol is unwieldy, which fails a basic efficiency test.

Our complete protocol addresses ((i.)) by separating the function of a decryption authority, which holds the secret key material that allows record decryption, from that of the court, which approves decryption requests. This separation of duties allows each decryption authority to confirm with the auditor that an audit record pertaining

[7]Secrecy from the court could also be achieved without per-record keys by double-encrypting the records, once under $\mathsf{pk}_{*,C}$ and once under another public key $\mathsf{pk}_{*,I}$ specific to the investigator.

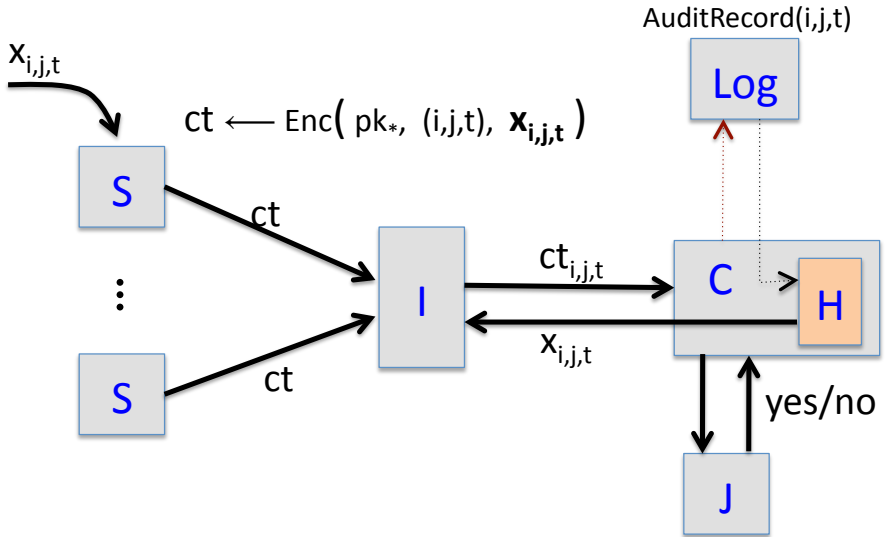

Figure 4.1: An insufficient approach which trusts the Court never to misbehave, shown with the court fortified by a hardware security module.

to the request in question has been recorded. Using a combination of secret sharing and threshold cryptography, we can instantiate our protocol so as to limit the trusted base to the requirement that t of N authorities are honest.

We address ((ii.)) by designing a new notion (auditable oblivious transfer, described in Section 4.3.1) and a protocol realizing it (our main protocol, described in Section 4.4) that guarantees that interactions between the court and the new authorities are subsequently auditable by a third party.

Finally, we address ((iii.)) by a straightforward application of *identity-based encryption* (IBE), which is public-key encryption but where the public keys are arbitrary strings, in our case the record tag (i, j, t) [332]. IBE allows us to compress the many per-record keys of Protocol 1 onto a single master key pair and then extract record-specific keys using the identity of a record later, should access to a record become necessary, greatly improving the deployability of a cryptographic solution to this problem. We use the IBE of Boneh and Boyen (BB-IBE) [60] with privacy for the investigator's requests provided by the blind key extraction protocol of Green

and Hohenberger [182] (introduced in Chapter 3, Section 3.2.2) modified to enable accountability as described in Section 4.3.1.

We outline our complete protocol, developed in more detail in Section 4.4 and shown in Figure 4.2, here. Chapter 3, Section 3.2 describes the various necessary primitives.

Full Protocol Description (Informal)

Setup. All parties generate key pairs for channel authentication and exchange public keys. Generate a master key pair $(\mathsf{mpk}, \mathsf{msk})$ as well as public parameters for BB-IBE. Split the IBE secret key into shares, msk_k, giving one to each decryption authority D_k. Further, the court and each decryption authority generate keys for a signature scheme.

Normal Operation. At the end of each time interval t, each data source S_i will encrypt any records about subject j generated during that interval under the master IBE public key, mpk, using each record's tag (i, j, t) as the required identity. This yields one ciphertext per source-user pair per time interval. These ciphertexts are sent (or made available) to the investigator I.

Investigator Query. When the investigator wishes to access the record for a particular tag $\mathsf{id} = (i, j, t)$, it presents this request to the court. The court records the fact and contents of the request with the Auditor. If the court grants the request, it issues an order approving decryption to each decryption authority and to the auditor. The decryption authorities confirm with the auditor that the order has been recorded, and then engage in accountable blind IBE extraction with the court. From blind extraction, each decryption authority receives an audit record containing a commitment to the id requested as well as an encryption under the Auditor's secret key of the randomness needed to open that commitment; the

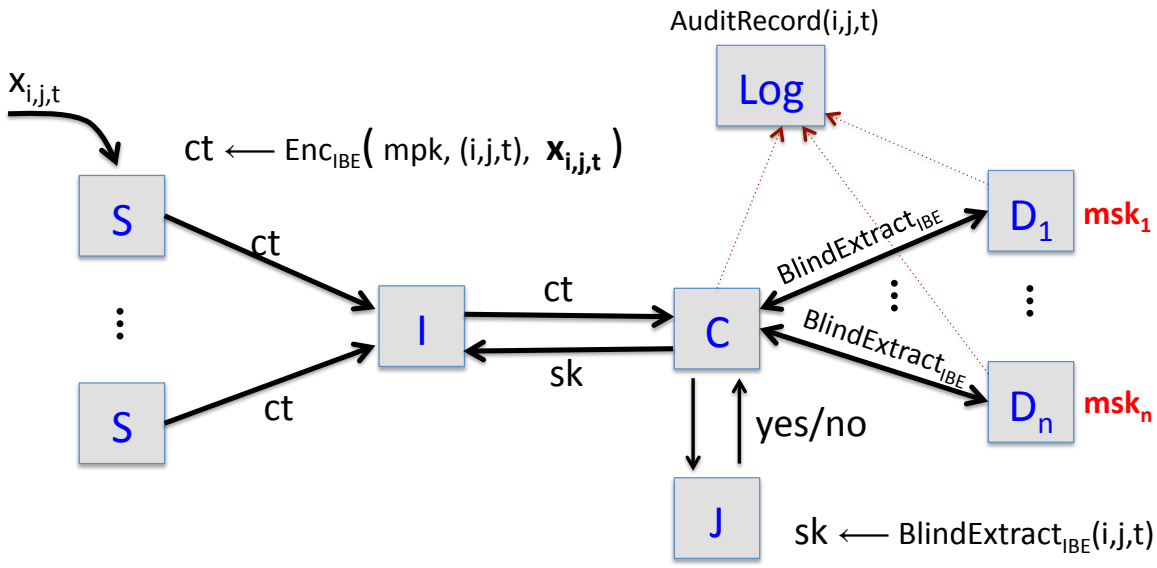

Figure 4.2: A high-level view of our protocol, showing the parties and their interaction.

court receives shares $\mathsf{sk}_{\mathsf{id}k}$ of $\mathsf{sk}_{\mathsf{id}}$. Each decryption authority records their audit record with the auditor. Finally, the court combines the received key shares to yield $\mathsf{sk}_{\mathsf{id}}$, which it then returns to the investigator, which decrypts the desired record.

4.3 Protocol-Specific Primitives

In this section, we describe primitives specific to our protocol for compelled data access, specifically focusing on the development of our new notion of *auditable threshold oblivious transfer* (aOT), which describes the functionality of decryption authorities D_k who are not cleared to see the investigator's requests while still allowing for the accountability properties described in Section 4.2.2. aOT provides a useful abstraction for explaining the guarantees provided by our entire protocol. Along the way, we define other necessary protocol-specific primitives. A general overview of the underlying cryptographic tools we employ can be found in Chapter 3, Section 3.2.

4.3.1 Auditable Oblivious Transfer

Oblivious Transfer

Oblivious transfer was defined and surveyed in Chapter 3, Section 3.2.2. Blind IBE extraction can be implemented via oblivious transfer, where we think of the holder of the master IBE secret key msk as a sender $\mathcal{S}$ holding messages $\mathsf{sk}_{\mathsf{id}}$ indexed by identifiers id, queried by a receiver $\mathcal{R}$ who wants to extract a key for a particular identity id [182]. Our protocol uses a threshold version of blind IBE extraction to provide for decryption authorities who hold (shares of) a secret key separately from the court, but who are not cleared to see the identifiers for which they are extracting keys.

Auditable Oblivious Transfer

Formulating blind IBE extraction as an oblivious transfer does not meet our needs, however, as no party or observer can prove that the key extracted via the protocol corresponds to the index authorized by the court. We therefore reformulate the concept of oblivious transfer into a more suitable notion for our purposes, *auditable oblivious transfer* (aOT), and give a construction by modifying an existing OT protocol. In aOT, the participants specify a passive auditor party $\mathcal{A}$, who will receive encrypted audit records from the protocol but does not directly participate.

Specifically, in aOT, a trusted setup party $\mathcal{T}$ generates a key pair $(\mathsf{pk}_{\mathcal{A}}, \mathsf{sk}_{\mathcal{A}})$ for a public-key encryption scheme and a key pair $(\mathsf{vk}_{\mathcal{R}}, \mathsf{sk}_{\mathcal{R}})$ for a digital signature scheme, dealing them as required below.

We specify the inputs and outputs to aOT as:

inputs: $\mathcal{S}$ takes $S = \{m_1, \ldots, m_s\}$ as well as $\mathsf{vk}_{\mathcal{R}}$; $\mathcal{R}$ takes $\ell \in \{1, \ldots, s\}, (\mathsf{sk}_{\mathcal{R}}, \mathsf{pk}_{\mathcal{A}})$

outputs: $\mathcal{R}$ receives m_ℓ, and $\mathcal{S}$ receives $(\mathsf{ct}, \mathsf{Sign}(\mathsf{ct}, \mathsf{sk}_{\mathcal{R}}))$ where $\mathsf{ct} := \mathsf{Enc}(\mathsf{pk}_{\mathcal{A}}, \ell)$, as well as a zero-knowledge proof $\Pi_{\mathcal{A}}$ that ct is a valid encryption of ℓ.

In particular, unlike vanilla OT, the sender receives output from aOT, namely an audit record containing an encryption of the index queried (under the auditor's public key $\mathsf{pk}_{\mathcal{A}}$, so the sender does not learn the index queried) and a signature on that ciphertext (under the receiver's signing key $\mathsf{sk}_{\mathcal{R}}$, to prevent its repudiating the audit record). The audit record also contains a zero-knowledge proof that this ciphertext is a valid encryption of the index queried in the protocol. The public-key encryption scheme used to create the encrypted audit entries should be chosen so as to make $\Pi_{\mathcal{A}}$ efficient.

The notion of aOT described here provides online auditability. Specifically, sender will detect if the zero-knowledge proof in the audit record fails, and can thus verify that the audit record contains the index requested by the receiver. In many scenarios, however, it suffices that we provide *offline* auditability, that is, a misbehaving receiver can be detected after-the-fact. Informally, at the conclusion of an *offline aOT* protocol, the sender obtains an audit record which enables the auditor to obtain the index that was requested (if the receiver is playing honestly) or to demonstrate cryptographically that the receiver was misbehaving.

4.3.2 Sharing the IBE master secret

Our protocol also requires that the IBE master secret key msk be distributed among a number of decryption authorities, a threshold group of whom are required for key recovery and decryption. Specifically, we must perform IBE setup and key distribution in a way that enables the accountable blind IBE extraction protocols mentioned above, ideally without requiring a special trusted party to execute the setup operations. This can be achieved by having each party generate their share of the master secret key and publishing the corresponding public key, then combining these to obtain public parameters using the techniques of Gennaro et al. [163].

Similarly, we must extend blind IBE extraction to support threshold extraction and key reconstruction, which can be done straightforwardly using a linear secret sharing scheme such as that of Shamir. This was described in Chapter 3, Section 3.2.4.

Auditable Threshold Oblivious Transfer

Armed with these definitions, we can now describe the complete *auditable threshold oblivious transfer* (threshold aOT), which describes the critical properties of our protocol.

Auditable threshold oblivious transfer is a protocol between a receiver $\mathcal{R}$, an auditor $\mathcal{A}$, and a subset of a tuple of active senders $W \subseteq (\mathcal{S}_1, \ldots, \mathcal{S}_n)$. For simplicity, we describe it informally here as an abstract multiparty functionality.

First, during a setup phase, a trusted party $\mathcal{T}$ generates two key pairs for a public key encryption scheme, $(\mathsf{pk}_*, \mathsf{sk}_*)$ and $(\mathsf{pk}_\mathcal{A}, \mathsf{sk}_\mathcal{A})$ as well as a key pair for a digital signature scheme, $(\mathsf{vk}_\mathcal{R}, \mathsf{sk}_\mathcal{R})$. $\mathcal{T}$ also generates shares of the secret key sk_*, $\mathsf{sk}_*^{(1)}, \ldots, \mathsf{sk}_*^{(n)}$, and distributes each $\mathsf{sk}_*^{(i)}$ to the corresponding sender $\mathcal{S}_i$.

Let $M = \{m_\ell\}_{\ell \in I}$ be a set of messages indexed by some set I. Define $C = \{\mathsf{ct}_\ell := \mathsf{Enc}(\mathsf{pk}_*, \ell, m_\ell)\}_{\ell \in I}$ to be the encryption of all messages in M under pk_*.

Now, we can describe threshold aOT as a functionality computed by $\mathcal{R} \cup \big(W \subseteq \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}\big)$ by the relation:

inputs: $\mathcal{R}$ is given $(C, \mathsf{pk}_*, \mathsf{sk}_\mathcal{R}, \ell \in I)$, each $\mathcal{S}_i$ is given $(\mathsf{sk}_*^{(i)}, \mathsf{vk}_\mathcal{R})$.

outputs: $\mathcal{R}$ obtains m_ℓ and each $\mathcal{S}_i$ obtains an audit record $(\mathsf{ct}, \mathsf{Sign}(\mathsf{ct}, \mathsf{sk}_\mathcal{R}))$ where $\mathsf{ct} := \mathsf{Enc}(\ell, \mathsf{pk}_\mathcal{A})$ as well as a zero-knowledge proof $\Pi_\mathcal{A}$ that ct is an encryption of ℓ.

If all senders $\mathcal{S}_1, \ldots, \mathcal{S}_n$ collude, they learn nothing from the protocol beyond their respective audit records, which in particular only disclose the index ℓ queried by $\mathcal{R}$ to the auditor $\mathcal{A}$. And if $\mathcal{R}$ colludes with a sub-threshold set of senders, the conspiracy

learns nothing about the decryption of any ciphertext $\mathsf{ct}_{\ell'} \in C, \ell' \neq \ell$. Finally, no conspiracy of senders can learn anything about the queried index ℓ.

As with aOT, we can similarly define the notion of offline auditable threshold oblivious transfer, where it suffices that an after-the-fact audit can identify a misbehaving receiver. Our protocol, described in Section 4.4, can be viewed as an offline threshold aOT between a sender comprising the auditor and all decryption authorities and a receiver comprising the court and the investigator.

4.4 Protocol for Accountable Compelled Access

This section presents our full protocol, elaborating on the from Section 4.2.4. The interaction between parties in the protocol is shown in Figure 4.2.

Protocol 2 (Formal).

Setup. The system is initialized as follows:

(i.) The decryption authorities D_k, $k \in \{1, \ldots, n\}$ run $\mathsf{Setup}_{\text{IBE}}$ using the distributed protocols from Gennaro et al. [163] to generate public parameters and master keys for the threshold BB-IBE scheme. Specifically, the distributed key generation protocol is needed to generate a tuple (g, g_1, α) in:

$$\mathsf{mpk} \; : \; \Big(g, g_1 := g^{\alpha}, h, v := e(g_1, g_2)\Big) \quad \mathsf{msk} := g_2^{\alpha}$$

Publish mpk to all parties. Each authority D_k also publishes to all parties the value $v_k = e(g_1^k, g_2)$ which corresponds to their share msk_i of msk, along with a non-interactive zero knowledge proof of knowledge of the corresponding msk_i.

(ii.) The auditor $\mathcal{A}$ generates a key pair $(\mathsf{pk}_{\mathcal{A}}, \mathsf{sk}_{\mathcal{A}})$ for a public key encryption scheme and a separate signing and verification key pair for a digital signature scheme.

The auditor retains $\mathsf{sk}_{\mathcal{A}}$ and the signing key, but publishes $\mathsf{pk}_{\mathcal{A}}$ and the verification key to all parties.

(iii.) The court C and each decryption authority D_k generate key pairs for a digital signature scheme and publish the verification keys to all parties: $(\mathsf{pk}_{\mathsf{C}}, \mathsf{sk}_{\mathsf{C}}), \{(\mathsf{pk}_{\mathsf{D}_k}, \mathsf{sk}_{\mathsf{D}_k}\})\}$.

(iv.) All parties, including the data sources S_i, generate separate channel authentication keys for each channel on which they must communicate and publish the corresponding verification key to all parties. Using TLS, these are just auxiliary signing and verification key pairs used to authenticate handshake messages when the channel is opened.

(v.) The auditor $\mathcal{A}$ maintains a counter ctr, initially 0 and incremented after each decryption transaction.

Normal Operation of the system involves encrypting records at time intervals t.

(i.) After each interval t, each data source S_i computes:

$$\begin{aligned} \mathsf{ct}_1 &:= \mathsf{Enc}_{\text{IBE}}\left(\mathsf{mpk}, (i,j,t), r_{i,j,t}\right) \\ \mathsf{ct}_2 &:= \mathsf{Enc}\left(H\left(r_{i,j,t}\right), x_{i,j,t}\right) \end{aligned}$$

Where $r_{i,j,t}$ is a random element of $\mathbb{G}_T$ chosen by S_i and H is a public hash function modeled as a random oracle. This technique, which we refer to as *hybrid encryption*, comes directly from Green and Hohenberger [182] and is useful for bounding the size of the record which must be encrypted using $\mathsf{Enc}_{\text{IBE}}$.

(ii.) S_i sends (or makes available) the ciphertext $\mathsf{ct}_{i,j,t} := (\mathsf{ct}_1, \mathsf{ct}_2)$ to I.

Investigator Queries. Over a secure channel, I can send signed requests to the court C for access to the record with a particular tag $\mathsf{id} = (i,j,t)$. These requests are

recorded by the auditor $\mathcal{A}$. If C approves the request, it issues a legal order and takes the following actions:

(i.) C reads the current state of the counter ctr at $\mathcal{A}$.

(ii.) The court then issues an *order*, $\mathsf{ct}_{\mathrm{C}} := \mathsf{Enc}(\mathsf{pk}_{\mathcal{A}}, \mathsf{ctr}, \mathsf{id})$, and sends a signed copy to the auditor $\mathcal{A}$. We denote by $\mathsf{rand}_{\mathrm{C}}$ the randomness used in generating ct_{C}.

(iii.) The court then sends the order and the value of ctr to each of the decryption authorities, $\mathsf{D}_1, \ldots, \mathsf{D}_n$ in turn, and initiates the threshold blind IBE extraction protocol described in Chapter 3, Section 3.2.2. In addition to this protocol, the court also proves to each authority D_k in zero knowledge that it knows how to unblind the value id it committed to in the blinded IBE extraction request, providing $\Pi_{\mathsf{ct}_{\mathrm{C}}}$. In addition to the syntactic checks required by threshold blind IBE extraction, D_k validates the syntactic correctness of the order (i.e., that the signature is valid, that the blind extraction request is acceptable), aborting if these checks fail. This allows the court to obtain a blinded share $\mathsf{sk}_{\mathsf{id}}^{(k)}$ of $\mathsf{sk}_{\mathsf{id}}$. D_k receives from this auditable oblivious transfer protocol an audit record A, which it submits to $\mathcal{A}$ along with the order and proof $\Pi_{\mathsf{ct}_{\mathrm{C}}}$ it received from the court, all digitally signed. Specifically, each D_k submits (τ, σ) where $\tau := \{A, \mathsf{ct}_{\mathrm{C}}, \Pi_{\mathsf{ct}_{\mathrm{C}}}\}, \sigma := \mathsf{Sign}(\mathsf{pk}_{\mathsf{D}_k}, \tau)$. The audit record in this case contains the id and randomness needed to open the commitment in the blinded IBE request, encrypted under the auditor's public key, $A := \mathsf{Enc}(\mathsf{pk}_{\mathcal{A}}, \{\mathsf{id}, \mathsf{rand}_{\mathrm{C}}\})$.

(iv.) Once the court has all n blinded shares of $\mathsf{sk}_{\mathsf{id}}$, it combines them to obtain $\mathsf{sk}_{\mathsf{id}}$, and returns $\mathsf{sk}_{\mathsf{id}}$ to I.

(v.) The investigator decrypts $x_{i,j,t} = \mathsf{Dec}_{\mathrm{IBE}}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}_{\mathsf{id}})$.

This protocol guarantees the offline accountability of all parties: any misbehavior can be detected by an overseer using the audit record. It is straightforward (and only modestly less efficient) to extend the protocol so that it provides online accountability,

in the sense that every party will always detect misbehavior immediately and can abort the protocol. Online accountability can be achieved if the court includes an additional zero-knowledge proof in its blind decryption request, proving that the blinded id in its decryption request is consistent with the encrypted id in the audit record. To facilitate these zero-knowledge proofs, one could use the CHK encryption scheme [61,80] to encrypt the entries in the audit log. By using the CHK scheme (with BB-IBE), these additional zero-knowledge proofs amount to "proving relations in the exponent," and can be done using standard extensions of Schnorr proofs [105, 329].

4.5 Prototype Implementation

We implemented the protocol of Section 4.4. Our implementation work can be broken down into two primary pieces corresponding to the two operational phases of Protocol 2: a component for encrypting records during ordinary operation, and a set of components for handling investigator queries, each component taking the role of one of the parties shown in Figure 4.2. We present benchmarks for both pieces in Section 4.6. We used a 128-bit security level across the board in our implementation; consistent with this, we use a 289-bit MNT elliptic curve for our algebraic and pairing operations and 3072-bit RSA signatures and hybrid public-key encryption.[8]

The encryption portion of our implementation consists largely of a novel implementation of the Boneh-Boyen identity based encryption (BB-IBE) scheme [60], based on the PBC library [256] for elliptic-curve and pairing support, the GNU Multiple Precision Arithmetic Library (GMP) for large number support, and PolarSSL for basic cryptographic primitives, including RSA. Our implementation uses hybrid encryption with AES/GCM at the 128-bit security level for both BB-IBE and standard public-key encryption. We used SHA-256 to derive an identity for BB-IBE from a

[8]It would be straightforward to extend this to 256-bit security, at the cost of approximately a 2x slowdown in our IBE primitives (the main computational bottleneck) according to our initial measurements.

suitably encoded form of the record tag (i, j, t). This portion of our implementation required approximately 7,800 lines of C++ code.

Our implementation supports investigator queries using a constellation of party binaries that link against the core cryptographic component described above. These party scripts use PolarSSL to secure mutually authenticated TLS channels between all parties (each party has two public/private key pairs, one for encryption and one for message signing and channel authentication, with keys distributed manually as per our trusted key exchange assumption). Together, these binaries required less than 1,000 additional lines of C++.

4.5.1 Deployment Concerns

Any real deployment of our protocol will have to reckon with complex deployment issues that determine its effectiveness. The goal of our proof-of-concept implementation is to demonstrate the viability and performance of our core approach, so it does not address these interesting problems. However, we summarize some of the more interesting deployment concerns here and consider how to address them.

Imprecise Record Tags

In some applications, the record tag may not be a clean set of indices (i, j, t) as we have described, but rather something like a person's name or address, which could have many forms or be subject to misspelling. Because our protocol uses the tag as an IBE identity, any variation in the tag will lead to the underlying records only being decryptable by different keys. Hence, our protocol inherently requires that tags be *canonicalized* prior to encryption. Ideally, this canonicalization can be done in a standard, publicly verifiable way that does not require the sharing of metadata between data sources and the investigator. Verifiability is important to ensure during an audit that the canonicalization was performed properly and cannot be abused by

the investigator to gain access to a different set of records than the court believes it is unsealing. One approach would be for all parties to agree ahead of time on a particular *locality-sensitive hash function* [325].

Many important application scenarios do not require this type of clustering of record tags, such as applications where our protocol is used to protect access to telephony metadata or cloud service provider business records, both of which naturally give robust identifiers to all data subjects.

Multiple Investigative Authorities

It is interesting to consider how our protocol applies in the case of multiple, collaborating investigators, such as might exist in the context of a law enforcement investigation spanning many jurisdictions or agencies. Certainly, a court can grant an order for any investigative body it recognizes. Our protocol supports this in a direct way—many different investigators can receive ciphertext from data sources (or ciphertext can be held either by a special custodian party or simply by the data sources themselves and be queried as necessary) and make requests of the court, with the source of each request recorded by the auditor.

More interesting is the case where multiple investigators collaborate and must share information, but we wish to constrain this sharing and guarantee its accountability. Our protocol only allows tracking of authorization to access information via an order; analysis of what happens to that information after it is disclosed, while an interesting, difficult, and well studied problem, is beyond the scope of this work.

Non-localized record analysis

Our algorithm focuses on the accountability of information *disclosure* via a court order, leaving aside the rich and extensive literature on what sorts of analysis can be performed *without disclosing data* in the first place. Our protocol is not suited to

most kinds of non-local analysis of compelled data (i.e., analysis which considers many records, as opposed to analysis of the records for a particular individual). Under our protocol, each individual record to be included in the broad analysis would require a separate order, with a separate investigator query to generate a per-record decryption key.[9] While we take no position on the advisability or utility of such practices, they are known to be of interest in real law-enforcement scenarios.[10]

Without summarizing the vast literature describing computation on encrypted data, we remark briefly on work that touches specifically on the problem we set forth in this work, namely accountable compelled analysis of data records by an investigator subject to restrictions and oversight by a court. Segal, Ford, and Feigenbaum propose using a private set intersection protocol to accountably determine co-located cell phones using so-called cell tower "dumps", or logs of all devices which connected to a certain antenna in a certain time period [330]. Kamara similarly suggests using a combination of structured encryption and secure function evaluation in a protocol called MetaCrypt [221], although MetaCrypt's graph-based approach to authorization makes its focus and applicability somewhat narrower than ours. Earlier work by Kamara [220] suggests using Private Information Retrieval techniques to achieve the same goals. Recent work by Bates et al. [30] addresses the problem of accountable wiretapping, but does so with a very wiretapping-specific model that does not facilitate accountability for facts beyond whether an interception happened during an approved window. Bates et al. also use purely symmetric cryptography, meaning that many parties to their protocol must hold and protect long-term secrets. Also, Bates et al. do not describe what accountability means in their model; this is not

[9]Decryption per-record could be avoided by mandating that sources generate a pseudo-record that contains all of the required data and requesting decryption only of the pseudo-record.

[10]For example, it has been disclosed that the N.S.A. uses various analyses that operate on the "social graph" of communicating people. See for example James Risen and Laura Poitras, "N.S.A. Gathers Data on Social Connections of U.S. Citizens", The New York Times, 28 September 2013, `http://www.nytimes.com/2013/09/29/us/nsa-examines-social-networks-of-us-citizens.html`

uncommon—the term is often used imprecisely in computer science, as we survey in Chapter 2.

From an accountability perspective, the fact that our protocol does not support such non-local analysis is advantageous to an investigator that wishes to demonstrate that it only accesses records in a particularized way. This argument is the motivation for the similar work of Liu, Ryan, and Chen, who further suggest that side channel information in the length and number of entries of a fully encrypted audit log is a useful tool for public accountability [251]. Our approach provides this same side channel information in addition to its other publicly verifiable properties.

4.6 Evaluation

We benchmarked our implementation using synthetic data corresponding to a realistic deployment scenario: the protection of telephony metadata in a large country, such as the program operated by the United States National Security Agency disclosed by Edward Snowden.[11] Specifically, we assume that a small handful of carriers are recording metadata on approximately 500,000,000 telephone numbers; that the investigator seeks orders with a 1-day granularity per the direction of the FISA court that records be submitted to NSA on "an ongoing daily basis"; and that anyone can build a record tag (i, j, t) by selecting (i.) the name of a carrier i, (ii.) a phone number of interest j, and (iii.) the date t on which the metadata were generated.

[11] Under this program, the N.S.A. collected all "call detail records" on "an ongoing daily basis. See a summary of the program and the order from the Foreign Intelligence Surveillance Court authorizing it in Glenn Greenwald, "NSA collecting phone records of millions of Verizon customers", The Guardian, 6 June 2013, `http://www.nytimes.com/2013/09/29/us/nsa-examines-social-networks-of-us-citizens.html`. Although the Snowden disclosures only named Verizon Business Services as a source of metadata, the Director of National Intelligence later confirmed that the program is "broad in scope" (See `http://www.dni.gov/index.php/newsroom/press-releases/191-press-releases-2013/868-dni-statement-on-recent-unauthorized-disclosures-of-classified-information`). It is believed by experts to cover a large fraction of all American telephone carriers.

Our system operates in two major modes: (i.) An *encryption mode*, the mode of normal operation, in which data sources encrypt records; and (ii.) An *investigation mode*, in which the investigator requests the decryption of a specific record. We examine the performance of these modes in turn.

4.6.1 Encryption Benchmarks

Under normal system operation, each S_i will hybrid-encrypt records under a tag determined as described in Section 4.5. Table 4.1 gives the average time per record for the hybrid encryption of 500,000,000 records ranging from 1-16 kB each, along with a breakdown of the average times for IBE-encrypting an AES key and AES-encrypting the record itself. We further give the total number of CPU-hours required to encrypt the full database of 500,000,000 records, as well as an estimated number of hours for performing the same computation on one standard data center rack worth of computing hardware.[12] The encryption process is entirely parallelizable and, in the deployment scenario we consider, would be distributed among all telecommunications providers in a country, with each provider only responsible for encrypting the records of their own customers. These benchmarks show that encrypting the telephone metadata of all users in a large country at a granularity of once per day would cost only a few hours of computing time on modest infrastructure and is thus very much feasible using even our (un-optimized) prototype.

Our prototype requires a storage overhead of 936 bytes per record for hybrid encryption, yielding a total storage cost of just under 2GB per one million records when the underlying records are 1kB.

[12]We assume a standard 42-unit rack, with 16 cores per rack unit, giving 640 cores per rack. We believe that such a small investment in infrastructure is offset by the compensating gains in accountability, especially as the infrastructure would be distributed among many data sources.

These experiments were conducted on a cluster of about 550 cores divided across 52 machines running Linux 2.6.32 on 64-bit x86 chips running at between 2 and 3 GHz.

We observe that, at May 2015 prices, our benchmarks imply a cost of approximately \$0.10-\$0.13 per 1 million record encryptions on Amazon's Elastic Map Reduce cluster computing rental service, depending on the record size. Thus, encrypting the whole database of 500,000,000 identifers per day would cost between \$50 and \$65.

4.6.2 Investigation Benchmarks

When the court approves a request by the investigator, it must also engage in blind key extraction with all decryption authorities serially. Table 4.2 reports the end-to-end latency experienced by the investigator for configurations with 1, 2, 4, and 8 decryption authorities. We also report the bandwidth requirements between parties in Table 4.3.

The end-to-end timing results in Table 4.2 show clearly that it takes only a few seconds for our prototype to process a single decryption request. Even with 8 decryption authorities, the protocol completes in under 3 seconds. We believe this is acceptable for practical purposes since we would expect the court to spend more than a couple of seconds to review the investigator's request before executing the protocol. Thus, it is unlikely that our request handling protocol impedes the execution of orders. As expected, the IBE operations for blind extraction dominate the total cost of the protocol. For example, with 8 decryption authorities, the IBE operations constitute 80% of the total end-to-end processing time. Moreover, the computation time scales linearly with the number of decryption authorities. For instance, when we double the number of decryption authorities from 4 to 8, the time needed for the blind extraction portion of the protocol also increases by roughly a factor of 2. We found that standard crypto-

Record Size	Average time per record encryption (ms)	Average time per IBE encryption (ms)	Average time per AES encryption (ms)	CPU-Hours for 500,000,000 records	Time for parallel encryption (hours)	Storage required (GB/1M records)
1k	12.55	11.65	0.71	1742.7	2.72	1.83
2k	12.89	11.78	0.76	1790.5	2.80	2.78
4k	13.29	11.77	0.83	1845.6	2.88	4.69
8k	14.19	11.85	0.98	1971.2	3.08	8.50
16k	15.83	11.87	1.31	2198.2	3.43	16.13

Table 4.1: Benchmark encryption times and storage costs for our prototype implementation using hybrid encryption on records ranging in size from 1-16kB. We also show the number of CPU-hours required to encrypt a realistic-sized database, with 500,000,000 identities as well as the (estimated) total time the encryption would take in parallel on a single rack of machines.

Number of decryption authorities	Average end-to-end time for decryption (s)	Average total time for blind IBE operations (s)
1	1.22	0.79
2	1.28	0.87
4	1.69	1.28
8	2.69	2.17

Table 4.2: The average end-to-end time ($n = 10,000$; initial measurements of the distribution of request times show that variance is minimal) to process an order in our prototype, along with the average total time required for all blind IBE operations across all parties, shown for configurations with 1, 2, 4, and 8 decryption authorities.

Parties	Outbound Traffic (kB)	Inbound Traffic (kB)
(Investigator, Court)	1.45	1.00
(Court, Decryptor)	1.16	1.00
(Court, Log)	2.23	0.01
(Decryptor, Log)	1.54	2.24

Table 4.3: The amount of communication required for processing an order in our prototype. For a pair of parties (A, B), we count bytes sent from A to B as "outbound" traffic, and bytes sent from B to A as "inbound" traffic. When we write (Court, Decryptor), we denote the traffic from the court to *each* decryptor, and likewise for (Decryptor, Log). These measurements only include data sent at the application layer, and do not, in particular, include the extra bandwidth needed for TLS.

graphic operations (TLS handshakes, RSA signing/verification/encryption, hashing) added only minimal overhead to our implementation.

Our bandwidth measurements (shown in Table 4.3) demonstrate that our protocol is extremely bandwidth-efficient. Processing a decryption request never requires more than 3kB of total communication between any pair of parties. While the total bandwidth for the court and the log does increase linearly in the number of decryption authorities, even with 100 authorities the total communication required of each party is still less than half a megabyte. This shows that bandwidth is unlikely to be a

bottleneck in processing requests; rather, the expensive BB-IBE operations dominate the overall cost of the request handling protocol.

The timing experiments were arranged with all parties on different machines, with the machines connected by a rack-local network. Due to this proximity, we found that network delays were not a significant factor in measuring either the absolute end-to-end system operation time nor for the variance. Each of these machines was running 64-bit Ubuntu 12.04 with a quad-core Intel i3 processor running at 2.3 GHz and 8 GB of RAM.

4.7 Related Literature

Although we have already contextualized the design and construction of accountable systems in Chapter 2 and the necessary cryptographic primitives in Chapter 3, it is useful to give a very brief, very narrow review of literature related to the protocol and application scenario discussed in this chapter.

Compelled access to data by law enforcement is closely related to the well studied question of *key escrow* systems [34, 118, 241, 286], which have provoked a strong negative reaction from the technical community [1, 2, 52]. Particularly relevant is the controversial and failed Escrowed Encryption Standard (EES), which provided access to encrypted communications for law enforcement investigation purposes by leaking encrypted key material through a Law Enforcement Access Field (LEAF) [286]. The protocols in EES lacked sufficient authentication and allowed the leaking of key material more widely [52]. EES did not provide any standard for accountability or control of access to LEAF data, contributing to the technical community's opposition to the standard and its deployment; many scholars asserted that the existence of LEAF data presented an inherent risk to the overall security of communications

under a LEAF-accessible key [1], opposition which continues to similar proposals to this day [2].

Another closely related line of work uses IBE to provide various types of oblivious authorization [62, 115, 362]. De Cristofaro et al. [115] develop a generalized notion of Private Information Retrieval and Private Set Intersection that supports a native notion of access control. Boneh et al. [62] develop a primitive they call *public key encryption with keyword search* that implies a construction of IBE, which uses oblivious authorizations to determine whether a specified keyword is in a stream of encrypted data. The work of Waters et al. [362] on building a searchable encrypted audit log using IBE is somewhat closer to our work, in part because of their use of IBE to eliminate the need for long-term secrets held by the data sources. However, in their setting, their policy-setting party (the "audit escrow agent") holds the master IBE secret key, and so can compromise the security of the protocol if it misbehaves. Our protocol uses a combination of separation of concerns and threshold decryption to maintain end-to-end auditability. In particular, this is the major advantage of our new notion, aOT.

Section 4.5 summarized the closely related works of Segal et al. [330], Kamara [220, 221], Liu et al. [251], and Bates et al. [30] which are similar in their goals to our work but differ significantly in approach and scope. The most similar of these, the work of Liu et al., is focused on a kind of accountable key escrow for end-to-end encrypted systems and is orthogonal to our goal of building accountability protocols for cleartext data records. In contrast to Liu et al., we provide an implementation and benchmarks and use IBE to avoid the need for independent per-user keys and the concomitant key management issues. Liu et al. also focus heavily on formally modeling the correctness of their access policy and the invariants it provides, but they do not conceptualize accountability as enabling the context required by their protocol for oversight.

4.8 Discussion and Extensions

We have presented a general protocol that provides cryptographic guarantees facilitating accountability for the process of compelling access to data using legal orders such as warrants. Our protocol has many desirable properties:

(i.) Any request for data validly approved by the court will result in the decryption of the authorized record. In this way, the protocol also serves to enable a *proof of performance* for accountability.

(ii.) No record can be decrypted without the authorization of the court. The correctness of every decryption can be verified offline: misbehavior by any party will be detected after the fact by an overseer.

(iii.) A third party not participating in the protocol can verify these facts and learns side channel information, such as how many decryption requests have been made and how many orders have been issued.

(iv.) Only the investigator and the court learn the identities corresponding to decrypted records. Ciphertexts can be identified by IBE identities and need not be stored with sensitive metadata.

(v.) A privileged oversight party can later confirm which records were authorized for decryption and that these match the records actually decrypted.

(vi.) No single party or machine in the system can compromise the above properties.

(vii.) Key management is straightforward: data sources do not need to retain long-term secrets; the system's master secrets are only used in a threshold manner and are never reconstructed; IBE allows us to compress many record-specific keys onto a single master key pair.

In particular, our protocol can be used to enable effective oversight of compelled access by law enforcement to data held by communications service providers, a problem of

real-world interest, making that access *accountable* to the political, legal, and social context in which it operates.

4.8.1 Extensions

We sketch some interesting extensions to our protocol:

Using a mix of cleared and uncleared authorities In our protocol, the decryption authorities D_k do not know which records they are extracting identity key shares for. However, it may be desirable for some or all authorities to be authorized to know which records are decrypted. Our protocol generalizes easily to cover this case—the court simply omits the blinding mechanism from the IBE extraction protocol for the cleared authorities, while employing it for the uncleared authorities. This results in a simpler and minimally more efficient protocol for the cleared authorities.

Access control within record time intervals Our setting, in which data sources encrypt all records $x_{i,j,t}$ for a particular user j in a time interval t under the same IBE identity (i, j, t), has the property that if *any* records in a time interval are disclosed, they will *all* be disclosed. We suggested in Section 4.2 that this can be mitigated by choosing short, granular time intervals (possibly even placing each record into its own interval). But this comes at the cost of requiring (possibly many) more requests by the investigator to cover large spans of time.[13] It is therefore natural to want access control within a single record interval. A simple solution is to encrypt records under the precise time T at which they are generated, using the record tag (i, j, T), then to re-encrypt these records under the broader record tag (i, j, t). Using this scheme, the investigator gains access to traffic analysis data for the entire interval t, but can be

[13] Also, to avoid traffic analysis, data sources must generate many dummy records when time intervals are short.

restricted to requesting specific record contents for particular times T. This extension is similar in setting but different in approach to the work of De Cristofaro et al. [115].

Control of escrowed key materials While we have described our protocol as protecting access to cleartext records such as communications metadata held by data sources, a straightforward application is to the case where these records themselves are the cryptographic keys used for end-to-end encryption by users of a communications service, possibly escrowed under a source-specific key $\mathsf{pk}_{\mathsf{S}_i}$ or a law enforcement key pk_{I} or stored with other metadata in a LEAF [118, 286]. Our protocol could be used to provide accountability for such a system. However, we do not suggest that such an application would be wise as a matter of policy—such systems have been heavily criticized as being inherently risky and a source of vulnerabilities [1, 2, 52]. We believe that our system is better suited to controlling the additional risks of compelled access in scenarios were data are already collected and retained.

4.8.2 On selecting Decryption Authorities

We conclude by remarking on the practical difficulty of selecting a trustworthy set of decryption authorities. While the selection of decryption authorities might be straightforward in some applications (e.g., when all authorities are cleared to see unblinded requests, as in the case where our protocol is used to protect escalations by support staff to access a user's escrowed key material which encrypts their cloud storage locker), the application of our protocol specifically and the notion of cryptographic accountability generally is difficult in the context of protecting access to records by law enforcement. It is necessary that all decryption authorities be trustworthy enough to reliably hold long-term secret shares without compromise, as re-keying and re-encrypting existing ciphertext may be prohibitively expensive. Further, it is necessary that the set of decryption authorities and overseers be expansive enough

that the political process governing the compelled access regime can reasonably believe that at least one competent authority will always object in the case that it detects misbehavior. These conflicting requirements come into even greater tension in settings where law enforcement access is multi-jurisdictional, especially if the jurisdictions span national borders.

While we cannot claim that a protocol such as ours completely solves the problem of making compelled data access accountable, we feel that our protocol provides interesting and useful knowledge about the space of possible designs for systems and processes that manage the oversight of compelled data access, a practice which is common, growing in importance for law enforcement, and increasingly a topic of public discussion. In particular, while we do not take a position on the rightness or legality of any particular data access regime, we do believe that any such regime should at least have strong accountability, providing strong evidence of any misbehavior while simultaneously guaranteeing, even to unprivileged observers, the enforcement of known policy requirements. Our protocol achieves such accountability.

Finally, our protocol is an excellent example of the technique of using cryptography to provide technical assurances that facilitate accountability when the policy to enforce is known and fixed. In the following chapters, Chapters 5 and 6, we turn to the question of how to use cryptography to facilitate accountability even when no policy can be decided on in advance.

Chapter 5

Constructing Accountable Algorithms

In this chapter, we turn our attention to what happens when no well-defined specification of acceptable behavior is available in advance of a system's development and deployment. In this case, traditional human-operated processes turn to oversight, or the nomination of some competent authority to determine the boundary between acceptable and unacceptable outcomes of a decision process, on a case-by-case basis. Usually, that authority also holds the power to demand under penalty of punishment the information required to review individual decisions. We propose that the same kind of oversight can be applied to the execution of automated decisions by computer systems: a designated oversight authority, empowered to demand information about individual decisions, can review those decisions for consistency with the law or with social or political values. To accomplish this, we need a way to bind the execution of a computer program closely to its inputs and outputs, ideally in a way that preserves legitimate secrecy. In this way, we can verify that the oversight authority can review the actual justification (in terms of policy and input) for the announced output (i.e., we can show publicly that the authority can review what actually happened

in a given decision). One way to achieve this is for the decision maker to give a zero-knowledge proof that it knows a full execution trace for the decision process using particular inputs and yielding particular outputs. The protocol described in this chapter combines this core technique of proving knowledge of a policy execution with cryptographic commitments to specify arguments and results. Our protocol enables designated authorities and the general public to review the specifics of decisions by an automated process, thereby providing robust oversight and accountability for decision makers.

We begin by describing a formal model of our setting in Section 5.1, and describe our protocol as parametrized over abstract primitives in Section 5.2. Section 5.3 describes a concrete realization of our protocol using primitives introduced in Chapter 3. Our implementation of this realization, its evaluation, and some example application scenarios are described in Chapter 6.

5.1 Setting and Model

We assume that an agency or authority A is making algorithmically mediated decisions about a group of N data subjects (denoted as $\mathsf{S}_1 \ldots \mathsf{S}_N$). These decisions are subject to oversight by an overseer O. In our tax audit example from Chapter 1, the tax authority plays the role of the authority A, each taxpayer plays the role of a subject S_i, and the role of the overseer O is played by a court, a legislative committee, or the tax authority's internal Inspector General's Office. We model each party as a computationally bounded process.

A chooses a policy, which comprises secret policy data $\hat{\mathbf{y}} \in \mathcal{Y}$[1] and a public function $f : \mathcal{X} \times \mathcal{Y} \times \{0,1\}^k \mapsto \{0,1\}^\ell$, operating on a subject-specific value $\hat{\mathbf{x}}_i \in \mathcal{X}$ and a suitably chosen random seed of k bits, and returns $\hat{\mathbf{z}}_i$, an ℓ-bit output which is the result of applying the policy for subject S_i using the given random seed. As an

[1] If the entire policy is public, $\hat{\mathbf{y}}$ may be nil.

example, a tax authority might adopt a policy saying "audit each taxpayer with a probability proportional to their total gross income". We assume that each subject can prove the correct value for his own subject-specific data $\hat{\mathbf{x}}_i$. (In our tax audit example, this corresponds to each subject asserting what values it submitted in its tax filing).

Any aspect of the policy that is public will be built into f, and $\hat{\mathbf{y}}$ will represent the non-public aspects of the policy. If A wishes to hold his function f oblivious to S_i, he may use a public function which can execute some class of functions $\mathcal{F}$ obliviously. In the limit, f would be an emulator for a general-purpose, Turing-equivalent RAM machine and a program to be executed by that machine would be incorporated into $\hat{\mathbf{y}}$. Previous work has shown how to use zk-SNARKs to build tolerably efficient oblivious computations for the class of Turing-equivalent functions [39, 42, 291]. However, generalizing over smaller classes of functions (thereby leaking partial information about A's function, namely to which family of functions $\mathcal{F}$ it belongs) can lead to significant gains in efficiency, as generalizing to Turing-equivalent RAM machines has considerable overhead.

An overriding design goal for our protocol is that the steps required of the S_is should not change when building an accountable automated decision process. Thus, our protocol is non-interactive and only ever requires optional verification steps from the S_is, O, or a curious member of the public. This design goes even as far as determining how the parties will communicate: we assume that the parties share a public log L for proofs and other public messages, which A can write into and any other player can read. We assume the contents of the log are authenticated, that the log is append-only, and that its contents are never in dispute. We also assume that A has an authenticated and confidential channel to send private messages to each S_i

and to O.[2] We also assume the existence of a random beacon that can produce a uniform random value $b_i \in \{0,1\}^k$ outside the control of any party.

The protocol must protect the confidentiality of $\hat{\mathbf{y}}$, which can be known only to A and O, and the confidentiality of each subject's private data $\hat{\mathbf{x}}_i$, which can be known only to A and S_i.

O's job is to make sure that the policy being used by A is acceptable, meaning that it is lawful and (if A is a government agency) consistent with public values. Determinations of legality and values-consistency can be difficult and time-consuming to make, so we do not assume that the set of acceptable policies is known or readily computable. Instead, we assume that O has access to an oracle $\mathcal{O}$ that determines the acceptability of A's policy, $\mathcal{O}(f, \hat{\mathbf{y}}) \to \{\texttt{accept}, \texttt{reject}\}$. Queries to this oracle are assumed to be very expensive and so can only be issued occasionally. (Real-life oversight requires the time and attention of a court, legislator, or high-ranking official, which is scarce, and might involve a lengthy investigation or deliberation.)

O can demand that A reveal $\hat{\mathbf{y}}$ and can punish A if A fails to reveal $\hat{\mathbf{y}}$, or if A's policy (i.e., the partial application $f(\cdot, \hat{\mathbf{y}}, \cdot)$) is unacceptable, or if A refuses to follow the protocol. We assume A will act to avoid this punishment. A has an approximate model of how the oracle will behave, and A will try to balance the risk of adopting an unacceptable policy against A's other legitimate objectives (such as the goal of deterring tax evasion, if A is a tax authority).

5.2 Accountable Algorithms: A General Protocol

We describe here a protocol for achieving accountable algorithms in the setting described above. Our protocol relies on a few abstract primitives: (i.) a zero-knowledge proof scheme $\mathcal{ZKP} = \{\mathsf{Prove}, \mathsf{Verify}\}$ amenable to protocols for verified computation;

[2] In practice, we anticipate that these channels might be realized via physical mail, as this requires no setup on the part of any decision subject.

function CommitEpoch($f, \hat{\mathbf{y}}$) ▷ At $\mathcal{T}_{\text{COMMIT}}$

$\rho \xleftarrow{R} \{0,1\}^k$, $(C_\rho, r_\rho) \leftarrow \mathsf{Commit}\,[\rho]$

$(C_{\hat{\mathbf{y}}}, r_{\hat{\mathbf{y}}}) \leftarrow \mathsf{Commit}\,[\hat{\mathbf{y}}]$

AuthenticatedWriteToLog

$(f, C_{\hat{\mathbf{y}}}, C_\rho)$

EndLog

end function

---

function RandomnessEpoch(b_i) ▷ At $\mathcal{T}_{\text{RANDOM}}$(Optional)

ReceiveFromPublicSource

$b_i \xleftarrow{R} \{0,1\}^k$

EndReceive

$(\mathsf{rand}_i, \pi_{\mathsf{rand}_i}) \leftarrow V_\rho(\hat{\mathbf{x}}_i \| b_i)$

$R = c_1 c_2 \cdots c_k \leftarrow G(\mathsf{rand}_i)$, $(C_R, r_R) \leftarrow \mathsf{Commit}\,[R]$

AuthenticatedWriteToLog

$(b_i, C_R, \pi_{\mathsf{rand}_i})$

EndLog

end function

---

function DecisionEpoch($f, \hat{\mathbf{y}}, \hat{\mathbf{x}}_i, \rho, b_i$) ▷ At $\mathcal{T}_{\text{DECISION}}$

$1.(C_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{x}}_i}) \leftarrow \mathsf{Commit}\,[\hat{\mathbf{x}}_i]$, $2.\hat{\mathbf{z}}_i \leftarrow f\,(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R)$, $3.(C_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{z}}_i}) \leftarrow \mathsf{Commit}\,[\hat{\mathbf{z}}_i]$

$$\Pi_i \leftarrow \mathsf{Prove}\left[\begin{array}{llllll} \multicolumn{6}{c}{\text{Given } (C_{\hat{\mathbf{y}}}, C_\rho, C_{\hat{\mathbf{x}}_i}, C_{\hat{\mathbf{z}}_i}),\ \text{the prover } \mathsf{A} \text{ knows a witness}} \\ \multicolumn{6}{c}{\mathbf{w} = (\hat{\mathbf{y}}, \hat{\mathbf{x}}_i, \hat{\mathbf{z}}_i, r_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{z}}_i}, \rho) \text{ such that}} \\ \multicolumn{6}{c}{\hat{\mathbf{z}}_i = f\,(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R) \text{ and}} \\ (C_{\hat{\mathbf{y}}}, r_{\hat{\mathbf{y}}}) & = & \mathsf{Commit}\,[\hat{\mathbf{y}}]\,, & (C_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{x}}_i}) & = & \mathsf{Commit}\,[\hat{\mathbf{x}}_i] \\ (C_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{z}}_i}) & = & \mathsf{Commit}\,[\hat{\mathbf{z}}_i]\,, & (C_\rho, r_\rho) & = & \mathsf{Commit}\,[\rho]\,, \\ (C_R, r_R) & = & \mathsf{Commit}\,[R]\,,^* & \text{and } \mathsf{Verify}\,[\pi_{\mathsf{rand}_i}] & = & 1.^* \end{array}\right]$$

AuthenticatedWriteToLog

$(C_{\hat{\mathbf{x}}_i}, C_{\hat{\mathbf{z}}_i}, \Pi_i)$

EndLog

SendAuthenticatedOverSecureChannel(S_i)

$(\hat{\mathbf{z}}_i, r_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{x}}_i})$

EndSend

end function

---

function OversightEpoch($\hat{\mathbf{y}}, r_{\hat{\mathbf{y}}}$) ▷ At $\mathcal{T}_{\text{OVERSIGHT}}$ (Optional)

SendAuthenticatedOverSecureChannel(O)

$(\hat{\mathbf{y}}, r_{\hat{\mathbf{y}}})$

EndSend

end function

Figure 5.1: The core accountable algorithms protocol, written as a set of randomized algorithms executed by the decision maker A, with one function per epoch. The statements designated by $*$ in the zero-knowledge proof are only included if the decision maker's policy is randomized and $\mathcal{T}_{\text{RANDOM}}$ occurred.

	Written by A into Log	A computes and knows secretly	Learned by each S_i over a confidential channel
$\mathcal{T}_{\text{COMMIT}}$	f $C_{\hat{\mathbf{y}}}$ C_ρ	$\hat{\mathbf{y}}, r_{\hat{\mathbf{y}}}$ ρ, r_ρ	
$\mathcal{T}_{\text{RANDOM}}*$	$(b, C_R, \pi_{\mathsf{rand}_i})$	$(\mathsf{rand}_i, \pi_{\mathsf{rand}_i}) = V_\rho(\hat{\mathbf{x}}_i \| b_i)$ $R \leftarrow G(\mathsf{rand}_i)$	
$\mathcal{T}_{\text{DECISION}}$	$C_{\hat{\mathbf{z}}_i}$ $C_{\hat{\mathbf{x}}_i}$ Π_i	$\hat{\mathbf{z}}_i = f(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R), r_{\hat{\mathbf{z}}_i}$ $\hat{\mathbf{x}}_i, r_{\hat{\mathbf{x}}_i}$	$\hat{\mathbf{z}}_i, r_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{x}}_i}$

Figure 5.2: The core accountable algorithms protocol proceeds in three epochs (shown here as rows): $\mathcal{T}_{\text{COMMIT}}$, $\mathcal{T}_{\text{RANDOM}}$, and $\mathcal{T}_{\text{DECISION}}$. In each epoch, the decision maker authenticates values by signing them under a key known to all parties and enters them into the log as shown. In $\mathcal{T}_{\text{RANDOM}}$ (which occurs only if the decision maker's policy is randomized), a uniform random value chosen beyond the control of any party, $b_i \xleftarrow{R} \{0,1\}^k$, becomes available to all parties by assumption; A authenticates this value and enters it into the log. In $\mathcal{T}_{\text{DECISION}}$, the prover A sends a decision and other values separately to each S_i over a confidential, authenticated channel as shown. An optional final epoch $\mathcal{T}_{\text{OVERSIGHT}}$, is not shown. During this epoch, the oversight body O may demand to learn A's secret policy and its commitment key $(\hat{\mathbf{y}}, r_{\hat{\mathbf{y}}})$.

(ii.) a secure commitment scheme, $\mathcal{C} = \{\mathsf{Commit}, \mathsf{VerifyCommitment}\}$; (iii.) a signature scheme $\mathcal{S} = \{\mathsf{Sign}, \mathsf{Verify}\}$; and (iv.) in the case the decision policy function f is randomized, a verifiable random function V and a pseudorandom generator G. These abstract primitives were developed in Chapter 3, signatures in Section 3.1.1, zero knowledge in Section 3.1.2, commitments in Section 3.3.1, and verifiable pseudorandomness in Section 3.3.3. We describe a realization of this protocol using concrete primitives in Section 5.3.

5.2.1 Protocol

Our protocol proceeds in four epochs, $\mathcal{T}_{\text{COMMIT}}$, $\mathcal{T}_{\text{RANDOM}}$, $\mathcal{T}_{\text{DECISION}}$, and an optional $\mathcal{T}_{\text{OVERSIGHT}}$. $\mathcal{T}_{\text{COMMIT}}$ and $\mathcal{T}_{\text{RANDOM}}$ happen before the time when the algorithmically mediated decision result $\hat{\mathbf{z}}_i$ is computed and $\mathcal{T}_{\text{DECISION}}$ happens at the time of the

decision. $\mathcal{T}_{\text{OVERSIGHT}}$, when it does happen, will happen after the decision has been announced to the affected subject. In each epoch, the decision making agency A computes values and publishes an authenticated (i.e., signed under a key known to all parties) copy of some of these values in a public, append-only log. For brevity, we often omit the authentication in the description below. In the final epoch, the oversight body, O, can demand to learn some of A's secrets. Figure 5.1, shows the protocol steps as algorithms executed by A. Figure 5.2 summarizes the state of knowledge of the parties and the values in the log in each epoch. We describe the steps taken by A in each epoch below:

$\mathcal{T}_{\text{COMMIT}}$

In $\mathcal{T}_{\text{COMMIT}}$, the authority A specifies its decision policy: it publishes a public function f as well as a commitment $C_{\hat{\mathbf{y}}}$ to $\hat{\mathbf{y}}$, comprising any secret portion of the policy which is common to all decision instances (e.g., trained weights for a machine learning classification model). A also generates and commits to a k-bit random key $\rho \leftarrow \{0,1\}^k$ for a VRF. The commitment function will generate random keys $r_{\hat{\mathbf{y}}}, r_\rho$ that can be used later to open and verify these commitments. Whenever A generates a commitment, A publishes the commitment into the log and retains the commitment key as a secret.

$\mathcal{T}_{\text{RANDOM}}$

If A's decision policy requires any random coins, they are generated during $\mathcal{T}_{\text{RANDOM}}$. Otherwise, the protocol moves ahead to $\mathcal{T}_{\text{DECISION}}$. We observe that, although the decision maker will always need random bits to issue commitments, we need not verify the unpredictability of these bits to the decision maker—the unpredictability of these bits protects the decision maker from information leaks, but as the decision maker could always simply disclose the information committed to, we do not consider

manipulation of these bits to be an attack and leave the decision maker free to acquire them however it wishes. Verifying the commitments does not require any random bits, as the random keys for the commitments will be supplied as inputs to the circuit. $\mathcal{T}_{\text{RANDOM}}$ and the activities described in this section apply specifically to random coins required by the decision policy function f.

In $\mathcal{T}_{\text{RANDOM}}$, a public random value $b_i \leftarrow \{0,1\}^k$ is revealed and entered into the log. By assumption, this value is drawn uniformly at random from $\{0,1\}^k$. We further assume that this value is uniquely identified with a particular S_i. Section 5.2.2 discusses the reasons for this assumption, ways this mapping might be achieved, and ways to ameliorate it.

In practice, such a value could be supplied by a random beacon [305] or from some public ceremony performed by A or by one or more other trusted entities on A's behalf [134].[3]

$$\mathcal{T}_{\text{RANDOM}}: \qquad b_i \xleftarrow{R} \{0,1\}^k$$

Using b_i, the decision maker A computes a seed rand_i for a pseudorandom generator G using his secret key ρ, the input of the decision instance under consideration $\hat{\mathbf{x}}_i$, and the trusted random value b_i revealed during $\mathcal{T}_{\text{RANDOM}}$. A computes this seed using a verifiable random function. A also computes all k of the random coins he will need using G to get $R = c_1 c_2 \cdots c_k$, commits to R, and records this commitment in the log.

$$(\mathsf{rand}_i, \pi_{\mathsf{rand}_i}) = V_\rho(\hat{\mathbf{x}}_i \| b_i)$$

Here, V_ρ is a verifiable random function keyed by the decision maker's secret key ρ used to digest these disparate values into a single seed. The decision maker also enters

[3] If the ceremony is performed by multiple entities, a fair randomness protocol (these are described in Chapter 3, Section 3.3.4) may be used to make trust in any single participating party sufficient. This technique allows many mutually distrustful parties to participate in generating b_i in such a way that most users would be able to trust at least one participant to be honest, and therefore to trust that b_i is in fact chosen at random. In this way, this step can be realized while adding only minimally to the trust requirements of the overall protocol.

into the log the proof of correct seed extraction π_{rand_i}. The string R contains all of the random coins needed by the evaluation of A's decision policy.

We again observe that the nature of the generator used to expand rand_i to R is not material to our analysis, so long as the function itself is disclosed. Any secure pseudorandom generator will work (and the generator itself need not be secret as rand_i is secret), but the oversight body must know which generator was used. Therefore, we require that A also record a copy of G in the log.

$\mathcal{T}_{\text{DECISION}}$

In $\mathcal{T}_{\text{DECISION}}$, the authority A computes several values for each subject S_i under consideration:

- $\hat{\mathbf{z}}_i = f\left(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R\right)$
- $(C_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{z}}_i}) = \mathsf{Commit}\left[\hat{\mathbf{z}}_i\right]$
- $(C_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{x}}_i}) = \mathsf{Commit}\left[\hat{\mathbf{x}}_i\right]$
- a proof Π_i, which is a zero-knowledge proof of the following NP statement (clauses in the statement marked $*$ are only included if the decision maker's policy is randomized and $\mathcal{T}_{\text{RANDOM}}$ has happened):

$$\Pi_i : \quad \mathsf{rand}_i = V_\rho(\hat{\mathbf{x}}_i \| b_i)$$

AND

Given $(C_{\hat{\mathbf{y}}}, C_\rho, C_{\hat{\mathbf{x}}_i}, C_{\hat{\mathbf{z}}_i})$, the prover A knows a witness

$\mathbf{w} = (\hat{\mathbf{y}}, \hat{\mathbf{x}}_i, \hat{\mathbf{z}}_i, r_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{z}}_i}, \rho)$ such that

$\hat{\mathbf{z}}_i = f\left(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R\right)$ and

$$\begin{aligned}
(C_{\hat{\mathbf{y}}}, r_{\hat{\mathbf{y}}}) &= \mathsf{Commit}\left[\hat{\mathbf{y}}\right], & (C_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{x}}_i}) &= \mathsf{Commit}\left[\hat{\mathbf{x}}_i\right], \\
(C_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{z}}_i}) &= \mathsf{Commit}\left[\hat{\mathbf{z}}_i\right], & (C_\rho, r_\rho) &= \mathsf{Commit}\left[\rho\right], \\
(C_R, r_R) &= \mathsf{Commit}\left[R\right],^* & \text{and } \mathsf{Verify}\left[\pi_{\mathsf{rand}_i}\right] &= 1.^*
\end{aligned}$$

For each value computed except rand_i (which A keeps secret), A authenticates the value and records it in the log:

$$\mathcal{T}_{\text{DECISION}}: \quad C_{\hat{\mathbf{z}}_i}, C_{\hat{\mathbf{x}}_i}, \Pi_i$$

Also during this stage, A sends $(\hat{\mathbf{z}}_i, r_{\hat{\mathbf{z}}_i}, r_{\hat{\mathbf{x}}_i})$ to S_i over a confidential authenticated channel. This allows S_i to verify the commitments to $\hat{\mathbf{z}}_i$ and $\hat{\mathbf{x}}_i$.

$\mathcal{T}_{\text{OVERSIGHT}}$ (Optional)

As an optional final step, the oversight body O can demand that the authority A send him the secret policy value $\hat{\mathbf{y}}$ and the key $r_{\hat{\mathbf{y}}}$ necessary to verify A's commitment $C_{\hat{\mathbf{y}}}$ to that value, previously published in the log. However, O may not demand the secret key ρ or the commitment keys $r_\rho, r_{\hat{\mathbf{x}}_i}, r_{\hat{\mathbf{z}}_i}$. ρ serves to blind the actual stream of pseudorandom bits used by A in its decision and therefore protects the privacy of $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{z}}_i$, the private data of each S_i.

During this epoch, a decision subject S_i who has discovered either that A's commitments $C_{\hat{\mathbf{z}}_i}, C_{\hat{\mathbf{x}}_i}$ were improper or that A used an incorrect $\hat{\mathbf{x}}_i$ for her decision may bring evidence of this fact to O, who can then punish A if A did indeed violate the protocol.

Also during this epoch, and possibly in response to such a challenge, O may elect to query the policy oracle, $\mathcal{O}(f, \hat{\mathbf{y}})$, to learn whether A's policy is acceptable. If A's policy is unacceptable or if any of A's commitments or proofs do not verify, O will punish A for its misbehavior.

5.2.2 Analysis

This protocol satisfies the above-described security model.

After the protocol, the actors learn the following:

S_i: Each subject S_i learns $\hat{\mathbf{z}}_i$, the fact that $\hat{\mathbf{z}}_i = f(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R)$, that rand_i was chosen uniformly and randomly beyond the control of any party, and that valid commitments to $C_{\hat{\mathbf{z}}_i}, C_{\hat{\mathbf{x}}_i}, C_{\hat{\mathbf{y}}}$, and C_ρ have been logged. S_i does not, however, learn the secret portion of the policy $\hat{\mathbf{y}}$.

O: The overseer learns the fact that $\hat{\mathbf{z}}_i = f(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R)$ for every $\mathsf{S}_i \in \{\mathsf{S}_1 \ldots \mathsf{S}_N\}$, and that each rand_i was chosen uniformly and randomly beyond the control of any party. O has the power to demand knowledge of the secret portion of the policy, $\hat{\mathbf{y}}$, which is common to all decisions. O also has the power to demand that A show that its commitment to this value was proper. O does not learn any $\hat{\mathbf{z}}_i$ or $\hat{\mathbf{x}}_i$, however, beyond what he can infer about them from the the information already described in combination with any information he can gain outside the protocol.

After the decision has been announced to each S_i, the agency A is bound to a specific choice of decision policy $f(\cdot, \hat{\mathbf{y}}, \cdot)$ as well as to a full explanation for each decision $\hat{\mathbf{z}}_i$. The secret portion of the policy $\hat{\mathbf{y}}$ can be reviewed separately by an oversight body O, which can be certain after the fact that the specific policy reviewed was in fact the one that was used for each decision. The oversight body can make this determination even without seeing subject-specific input that would compromise the privacy of the decision subjects.

Specifically, the oversight body can determine that A's overall policy $f(\cdot, \hat{\mathbf{y}}, \cdot)$ is proper and was used in determining each result $\hat{\mathbf{z}}_i$ *without* referencing the corresponding subject-specific input $\hat{\mathbf{x}}_i$ by verifying the published zero-knowledge proof Π_i, which ensures that the agency knows values that comport with the published commitments to each $\hat{\mathbf{x}}_i$ and to the corresponding $\hat{\mathbf{z}}_i$. The function f, which is used to generate $\hat{\mathbf{z}}_i$ from $\hat{\mathbf{y}}$ and each $\hat{\mathbf{x}}_i$, is public, so O can be sure it is not improper in some way (for example, f might contain a switch statement over the surname of a specific decision subject S_j which causes the function to misbehave only on S_j's input; because the

function is public, O can inspect it to rule out this kind of bad behavior). To the extent misbehavior in f exists and is triggered by improper values in A's secret input $\hat{\mathbf{y}}$, O has the power to compel A to reveal $\hat{\mathbf{y}}$ and $r_{\hat{\mathbf{y}}}$. O can then check that $\hat{\mathbf{y}}$ matches the previously published $C_{\hat{\mathbf{y}}}$, thereby tying the revealed value to the proof Π_j that was also provided to S_j. In this way, O can review all aspects of A's decision process *except* the confidential information in $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{z}}_i$. From this review, O can determine whether A is using an approved policy for each decision (because O can inspect the secret portion of the policy $\hat{\mathbf{y}}$ directly) and that this policy was actually the one used to produce each result $\hat{\mathbf{z}}_i$ (because this fact is entailed by the proof Π_i). Further, O can elect to consult the policy oracle $\mathcal{O}(f, \hat{\mathbf{y}})$ to learn whether A's policy is acceptable. If any decision is improper or a policy is unacceptable, O has the cryptographic evidence necessary to justify punishing A.

Simultaneously, each data subject S_i receives its result $\hat{\mathbf{z}}_i$ and $r_{\hat{\mathbf{z}}_i}$ and so can verify $C_{\hat{\mathbf{z}}_i}$. Having done this, each S_i can verify Π_i and learn that A is committed to a full explanation for its decision $\hat{\mathbf{z}}_i$. Even though she cannot see the full policy, she knows that O can and trusts that O will object and punish A if the policy is in any way lacking. Further, because each S_i knows its correct $\hat{\mathbf{x}}_i$ and has learned $r_{\hat{\mathbf{x}}_i}$, she can check the veracity of $C_{\hat{\mathbf{x}}_i}$ and thus knows whether the proper value of $\hat{\mathbf{x}}_i$ was used to produce $\hat{\mathbf{z}}_i$ from her verification of Π_i. If the value of $\hat{\mathbf{x}}_i$ used is not the correct $\hat{\mathbf{x}}_i$ for S_i, S_i can demonstrate to O that A has misbehaved by revealing all or part of the correct $\hat{\mathbf{x}}_i$, which is enough to demonstrate that the incorrect value was used in the decision. O can then punish A for behaving improperly or rule that the challenge is spurious.

Observe that even though S_i reveals her subject-specific input in this case, her result $\hat{\mathbf{z}}_i$ is not revealed to the extent that f depends on the random seed value provided to it, $\mathsf{rand}_i = V_\rho(\hat{\mathbf{x}}_i \| b_i)$. Because O is never given access to ρ, the randomization of A's policy effectively blinds the computation of the result, to the extent O cannot

predict it from the other inputs. It is an interesting deployment consideration the extent to which each subject's individual data and result should or must be available to the overseer.

A small but manageable complication to this analysis occurs when A's policy requires some random coins in order to be computed. A malicious decision maker who knows a subject's inputs can bias results by selecting the random choices it desires. This is true even if we replace the use of randomness inside the policy with randomness sourced from a pseudorandom generator, as a malicious decision maker can still influence the choice of seed. Because these choices will be indistinguishable from some actually random choice, this defeats all accountability provided by our protocol! In order to provide A with verifiable random bits, we assumed that those bits came from a trusted beacon or fair randomness protocol. We additionally stipulated that the trusted bits had to be used to generate a seed that was cryptographically bound to the subject under consideration S_i in a verifiable way. We accomplished this by using the concatenation of these strings as an argument to a verifiable random function V_ρ, keyed by a secret key chosen by the decision maker, ρ. Because A commits long in advance to his secret key ρ, the random coins used for any given subject S_i are unpredictable to him, hence he cannot use them to bias the decision. Without the assumption that the beacon value b_i is assigned to a specific subject S_i, our protocol would be subject to a kind of *randomness shopping* attack, in which a malicious decision maker uses random values from a trusted source such as a beacon, but uses discretion to assign those values to the individual S_i so as to bias particular outcomes $\hat{\mathbf{z}}_i$.

This assumption can be ameliorated in several ways. We could, for example, structure the trusted beacon so that it does not only produce random values b_i, but indexed tuples (i, b_i) and demand that the decision maker commit in $\mathcal{T}_{\text{COMMIT}}$ to a mapping $i \mapsto \mathsf{S}_i$. Alternatively, in situations where at least some portion of $\hat{\mathbf{x}}_i$ cannot

be predicted by A in advance and the min entropy of $\hat{\mathbf{x}}_i$ is high enough, we can simply eliminate the assumption of a trusted beacon and compute $(\mathsf{rand}_i, \pi_{\mathsf{rand}_i}) = V_\rho(\hat{\mathbf{x}}_i)$. This technique applies to our tax audit example, where even though the tax authority likely has a strong prior for all of each S_i's filing information, tax forms could easily include an extra field asking the filer for a random nonce or pin code. Indeed, forms in the United States already ask for a (low entropy) pin code to authenticate the filer for customer service. However, this technique would not apply well when the decision subjects do not have input or visibility into their data, such as in a consumer scoring application. We consider the proper choice of how to handle randomized decision policies an interesting deployment concern for our protocol.

5.2.3 Extensions

Ensuring properties of secret policies

Suppose a decision authority wishes to demonstrate some property $\mathfrak{P}$ of its policy above and beyond the policy compliance oversight provided by the oversight body publicly. For example, an authority might wish to demonstrate that a scoring algorithm used to determine credit eligibility or terms does not use race or gender as an input, even in the secret part of the decision policy. Let $C_{\mathfrak{P}} : \mathcal{F} \to (\mathcal{Y} \to \{1, \perp\})$ be a predicate checking function for $\mathfrak{P}$. That is, $C_{\mathfrak{P}}(f)(\hat{\mathbf{y}}) = 1$ if and only if A's policy satisfies the predicate $\mathfrak{P}$. Clearly, if $C_{\mathfrak{P}}(f)$ is independent of $\hat{\mathbf{y}}$, A can demonstrate publicly whether his policy satisfies $\mathfrak{P}$. If, however, the property $\mathfrak{P}$ cannot be determined from f alone, each S_i will need some additional information to which it does not ordinarily have access in order to determine if $\mathfrak{P}$ holds.

Fortunately, a simple extension to the above protocol can provide sufficient information to assure all parties that $\mathfrak{P}$ holds even when it depends on the secret portion of A's policy: we simply compute the partial application $g : \mathcal{Y} \longrightarrow \{0, 1\}$ as

$g(y) = C_{\mathfrak{P}}(f)$ and then augment the statement proved by the zero-knowledge proof to include the condition $g(\hat{\mathbf{y}}) = 1$.

Using this extension, we can augment the above protocol to assure each decision subject directly of the truth of any NP-computable predicate over the agency's policy, regardless of whether that predicate depends on the secret portion of the policy.

Minimum disclosure escalations

Above, we stated that, in order to demonstrate that an incorrect value of the subject-specific input was used for her case, each decision subject S_j for whom A used an incorrect subject-specific input $\hat{\mathbf{x}}'_j$ to obtain a result $\hat{\mathbf{z}}_j$ would need to reveal all or part of $\hat{\mathbf{x}}_j$ to O in order to demonstrate this fact. Here, we explain the escalation process more carefully. Consistent with our security goal of minimizing the disclosure of subject data to the oversight body, S_j must, at minimum, disclose enough for the overseer to establish two facts:

(i.) That $\hat{\mathbf{x}}_j$ is the correct subject input for S_j.

(ii.) That some value $\hat{\mathbf{x}}'_j \neq \hat{\mathbf{x}}_j$ was the one used in the decision $\hat{\mathbf{z}}_j$.

To establish the first of these, S_j must necessarily disclose $\hat{\mathbf{x}}_j$ so that the overseer can determine that the value is correct. But it is not, in general, necessary for the overseer to see all of $\hat{\mathbf{x}}_j$ to learn this: it is enough for S_j to be satisfied that the value used for $\hat{\mathbf{z}}_j$ is acceptable to her and to escalate if it is not. Therefore, a challenge really only requires the unblinding of information sufficient to establish only that *some portion* of the $\hat{\mathbf{x}}_j$ is incorrect. By carefully structuring the commitment the authority uses to $\hat{\mathbf{x}}_j$ in the accountability protocol, we can make it possible for S_j to unblind the minimum portion of the correct value to establish this fact. In particular, if $\hat{\mathbf{x}}_j$ is a vector and the authority commits to the dimensions of that vector using a Merkle tree [262], it is possible for S_j to prove that a value $\hat{\mathbf{x}}'_j \neq \hat{\mathbf{x}}_j$ was used by the agency by showing only a single incorrect dimension to the overseer.

5.3 Realizing Accountable Algorithms with Concrete Primitives

We realize the protocol of Section 5.2 using the following concrete primitives developed in Chapter 3: (i.) Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) (Section 3.3.2); (ii.) random oracle commitments (Section 3.3.1), realized efficiently for zk-SNARKs using a collision-resistant hash function due to Goldreich, Goldwasser, and Halevi (GGH) that relies on the hardness of certain lattice problems (Section 3.3.1); (iii.) if the decision policy is randomized, a verifiable unpredictable function due to Dodis and Yampolskiy which, using a procedure described below, allows the verifiable extraction of a seed for a pseudorandom generator (Section 3.3.3); and (iv.) also if the decision policy is randomized, as a pseudorandom generator (also Section 3.3.3), we use Keccak, the algorithm underlying the new SHA3 standard, in its SHAKE128 configuration.

Because the preprocessing zk-SNARK system we make use of to realize this protocol is *malleable* under certain conditions (in the sense that an adversary can tamper with a valid proof to make a valid proof of a different, related statement), authentication of proofs entered into the log is always necessary in order to attribute statements to the agency A. We achieve such authentication using RSA signatures in the standard way, using SHA-256 to prevent the signatures themselves from being mauled, as described in Chapter 3, Section 3.1.1.

We remark that we have specified the use of Dodis and Yampolskiy's verifiable *unpredictable* function rather than a true verifiable random function. This is because their VRF requires the computation of a pairing both to compute the function and to verify the proof it generates. Here, we argue that the VUF is sufficient for our purposes. First, observe that the output of the VUF is uniformly distributed over its range, namely all of $\mathbb{Z}_p$. However, represented as bit strings, these values show a bias;

namely, their high-order bits are not pseudorandom, as small elements of the field do not require the highest-order bits to represent and therefore bias these bits towards 0. As long as the space of field elements is large enough, however, a uniform element will require approximately p work to rediscover once it is fixed. Even an adversary who only seeks a collision will still require approximately $\sqrt{p}$ work. Because we are absorbing the output of the VUF into a sponge function, namely SHAKE128, all of this entropy is available to us for the purpose of quantifying the effort required to reverse engineer the state of the sponge. Since we operate in the group over which the SNARK QAP operates, which is 128 bits in size, we obtain the desired 128-bit security level. To be conservative and achieve the desired security level also against collisions, we can execute the VUF at two points, yielding two pseudorandom field elements, and absorb both into the sponge. In fact, our implementation takes this conservative approach.

An alternative argument for transforming the bits of the VUF to true pseudorandom bits comes from a standard *randomness extraction* technique. Observe that, while the high-order bits of the VUF are biased, the low order bits are not. Indeed, let $n \in \mathbb{N}$ be such that $2^n < p$. We compute the distance between the uniform distribution on $\{0,1\}^n$ and the distribution induced by taking the low-order n bits of the VUF output. This distance is naturally bounded by $2^n/p$. To see this, consider that if p were a perfect multiple of 2^n, then no bias would take place. However, since it is not, some nontrivial values of the form $x \mod 2^n$ occur slightly more often than they do in the uniform distribution. So at most, the statistical distance will grow by $2^n/p$. Since our $p \sim 2^128$, we can achieve very close to 128-bit security by keeping all but a constant number of low-order bits of the VUF output.[4] Again, we can amplify our work factor by evaluating the VUF at multiple points.

[4]We credit this argument to our collaborator from Chapter 4, David Wu, from personal communication.

Chapter 6

Example Accountable Algorithms

In this chapter, we describe our implementation of the protocol of Chapter 5 (Section 6.1) and describe the generation of constraint systems/circuits for functions necessary to use zk-SNARKs in that protocol (Section 6.1.1). We also describe a general evaluation of our implementation (Section 6.2). We finish by describing several concrete example circuits we have implemented and evaluated, and our early experiences trying to apply our protocol to real-world scenarios (Section 6.3).

6.1 Implementation

We implement a realization of the protocol of Chapter 5, Section 5.2 using the concrete primitives developed in Chapter 3 and mentioned in Chapter 5, Section 5.3. Our implementation relies on `libsnark`, the zk-SNARK library of Ben Sasson et al. [42] for computing and verifying proofs.[1] We found generating circuit representations of functions using the "gadget library" of `libsnark` somewhat challenging, however, so we developed our own suite of tools in the Go programming language for generating constraint systems based on our own, modified gadget abstraction. This

[1] `libsnark` is a library under active development, especially as it becomes a part of more and more projects both inside and outside the research community. The latest version may be found at the library's website, `https://github.com/scipr-lab/libsnark`.

system, which constitutes approximately 1000 lines of code, is described more fully in Section 6.1.1.

To integrate constraint systems generated by our tools with `libsnark`, we developed a small driver program of about 200 lines of C++ code, which invokes the correct routines within `libsnark` to implement the core cryptographic algorithms of KeyGen, Compute, and Verify. We use other scripts surrounding these to represent the other activities in $\mathcal{T}_{\text{COMMIT}}$, $\mathcal{T}_{\text{RANDOM}}$, and $\mathcal{T}_{\text{DECISION}}$—for example, to achieve commitments using the knapsack-based hash function of Goldreich, Goldwasser, and Halevi (GGH) [169], we wrote a separate integration tool that leverages the implementation of this construction within `libsnark` of approximately 350 of C++. For test purposes, we simply used the file system of our test machine to represent the log which is visible to all parties. In practice, one would instead use some kind of tamper evident log, either held by an auditor with updates published on a regular schedule, realizable efficiently using the scheme of Crosby and Wallach [109]; or implemented on top of a tamper-resistant system such as Bitcoin, as suggested by Clark and Essex [98].

To properly implement $\mathcal{T}_{\text{RANDOM}}$, and to generate random bits for use in commitments, we needed a pseudorandom generator. We realize this requirement using the *cryptographic sponge function* Keccak, now standardized as SHA-3 [45], for which there is an experimental package.[2] Similarly, we use RSA signatures implemented by the Go standard library, which implements the RSA de-facto standard of RSASSA-PSS,[3] which in turn implements the classic construction of Bellare and Rogaway [36]. We also implemented the verifiable random function of Dodis and Yampolskiy [127] in Go, borrowing group parameters from `libsnark` for ease of integration and validity of the resulting proofs of randomness.

[2]This package is available at `https://golang.org/x/crypto/sha3`.

[3]Though promulgated by RSA Security Inc. [216], this standard was later adopted by the IETF as RFC 3447.

Consistent with our goal of a 128-bit security level across the board, we use `libsnark` with its default Barreto-Naehrig curve, the SHAKE128 configuration of Keccak, and 3072-bit RSA keys for signatures.[4] For commitments, we use the knapsack-based hash function of Goldreich, Goldwasser, and Halevi as implemented inside `libsnark`. To achieve 128-bit security, we add 256 bits of randomness to our input, drawn from `/dev/urandom/` and mixed using the SHA-512 implementation from PolarSSL, for which we had existing code in our implementation of the protocol from Chapter 4.

6.1.1 Constraint System Generation

To generate constraint systems, we elected to develop our own tools rather than use either of the two "gadget libraries" that are part of `libsnark`. Both components are meant to allow the bottom-up manual coding of circuits from simple components which can then be easily extracted into constraint systems in the data structures needed for key and proof generation. In practice, however, the `libsnark` notion of a gadget is somewhat limiting. Primarily, this is because in a real-world scenario, the code for a function must be available both to the key generator for generating the evaluation and verification keys $(\mathrm{EK}_F, \mathrm{VK}_F)$ and to the prover, who must compute $\hat{\mathbf{z}}_i = f\left(\hat{\mathbf{x}}_i, \hat{\mathbf{y}}, R\right)$. However, because `libsnark` expresses circuits only as in-memory C++ data structures, these must be regenerated by each party's program. Evaluating the program in circuit form and extracting output is also somewhat unwieldy, as `libsnark`'s tools for generating satisfying assignments for constraint systems generated by the gadget libraries are rather primitive. Instead, we built our own gadget abstraction in the Go programming language.

[4] This is consistent with guidance from RSA Laboratories about key sizes: `http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/key-size.htm`. RSA maintains an active program to track the efficiency of widely known factoring algorithms and implementations against the work factor required to break particular instances of the RSA problem.

We view the function of a gadget as producing (i.) a constraint system suitable for proof generation, and (ii.) a function that maps inputs supplied by the prover to a satisfying assignment for the constraint system. Because of this, our gadgets differ from those in `libsnark` in that they offer both a GenConstraints functionality that outputs constraints suitable for use by the cryptographic components of `libsnark` *and* a Codegen functionality that outputs Go source code for a function that inputs to a higher-level description of a decision policy onto a satisfying assignment for the constraint system output by GenConstraints. These functions may also supply *nondeterminisitic advice*, since to compute a function the constraint system need only verify its correctness. Often, a problem which is simple to verify in the R1CS language used by `libsnark` is very expensive to compute directly as a circuit.[5]

All of this suggests to us that constraint programming is a ripe area for the development of an optimizing compiler. While other work has produced "constraint system compilers" [39,42,291], we remark that these programs have not, in general, been very sophisticated and mostly try to find the smallest constraint system for individual lexical elements, without any later optimization of the resulting constraint system or attempts at designing a programming model amenable to whole-program optimization. Later work by Costello et al. [104] reports large performance increases from such techniques along with many cryptographic improvements. Such a programming model approach is the focus of work by Stewart and Kroll on Snårkel,[6] which aims to bring the research techniques of the programming languages community to bear on these problems. In particular, Snårkel re-uses our `libsnark` integration tool to integrate constraint systems generated by our compiler. We use the Snårkel compiler to generate constraint systems for the examples described in Section 6.3.

[5] An easy example of this is division. In R1CS, a division circuit computing the inverse of a field element bit-wise is a very large object that can be verified in a single constraint. So computing that $a/b = c$ may cost many hundreds of constraints when the two constraints $b * c = a$ and $b * x = 1$, where x and c are supplied by the function produced by Codegen, will express the same computation.

[6] `https://github.com/gstew5/snarkl`

6.2 Evaluation

The performance of `libsnark` is already well studied and understood by its authors [39, 42]. Our implementation can be understood most simply as taking the circuit C_f for a function f which would be used with `libsnark` and adding some extra machinery by embedding it as a sub-circuit in a separate *accountability circuit*, $C_{\mathrm{AA},f}$, which augments C_f to include verification of the four commitments and, if necessary, the verifiable random function and associated fifth commitment used by the protocol of Chapter 5, Section 5.3.[7] For this reason, the primary goal of our benchmarking is to measure the overhead of $C_{\mathrm{AA},f}$ with respect to C_f. We remark that this overhead is constant, since the number and complexity of the commitments in our protocol does not depend on the complexity of f or its circuit representation.

Our accountability circuit $C_{\mathrm{AA},f}$ implementation requires verifying either three commitments (when no randomness is required) or four (when it is). Assuming we can pack committed values into single field elements, we can execute a commitment at 128-bit security using a single constraint.[8] This includes the addition of a random blinding value. If we need to pack a string of n bits into a field element, we can do this in n constraints. Thus, our accountability subcircuit requires at most $4n$ additional constraints, where $n \geq \max(|\hat{\mathbf{x}}_i|, |\hat{\mathbf{y}}|, |\rho|, |\hat{\mathbf{z}}_i|)$. We can approximate this as at most 1024 extra constraints. Our measurements confirm the published performance of `libsnark`of approximately 0.21ms of key generation time per constraint, 0.21ms of proving time per constraint, and therefore our subcircuit an additional 0.215 seconds of key generation and proving time ($\pm 0.1\%$, $n = 1000$).

We stress that our performance numbers are preliminary and based on limited experimentation. Further careful experiments are necessary to provide robust eval-

[7] Recall that verification of the VRF and a fifth commitment to pseudorandom bits required by f are only necessary if f is randomized.

[8] In specific, the knapsack-based hash of Goldreich, Goldwasser, and Halevi scales (very roughly) in security as $d|\mathbb{F}_p|$, where d is the dimension of the fixed public matrix over $\mathbb{F}_p$.

uations of performance using better-understood cryptographic primitives (such as widely used hash functions to replace our GGH hash function[9]

6.3 Examples

Here, we consider the space of applications for accountable algorithms. From a practical perspective, zk-SNARKs require significant computational resources and time for proving, even if they are extremely efficient to verify. Therefore, they are most applicable in situations where intense proving resources are not an issue or where the function to which they are to be applied can be represented as circuit or constraint system that is not too large. Such applications will, in general, be situations where proofs are generated infrequently and verified frequently, where few proofs are required, or where the decision maker involved can invest significant computational resources.[10]

This section details several concrete examples of classes of algorithm for which these techniques could be practical.

6.3.1 Linear Classification: Scoring and Risk Assessment

Very simple linear models, with some extra techniques layered on top, have been very successful in terms of practical use in classification tasks. This stems in part from their ease of training, but also from the fact that they produce robust and usable results. In this section, we consider the implementation of some core classification techniques as constraint systems usable within zk-SNARKs. Such systems are especially common

[9]While the GGH hash function suffices for our purposes, selecting parameters for it to meet a particular work factor requirement requires significant guesswork inappropriate for real deployment.

[10]In our tax auditing example from Chapter 1, let us assume that the tax authority would need to compute proofs for a nation-sized group of taxpayers once per year. For convenience, we will assume this means 500,000,000 proofs per year. Assuming standard 42-unit datacenter racks, with 40 units utilized per rack and 16 cores per unit, such an agency would need roughly one rack of servers per 40 seconds of proving time.

in systems that assign consumer scores such as credit and insurance underwriting scores, and in systems which assign risk for audit or fraud management purposes.

In general, a linear classifier takes the form:

$$y = f(\vec{w} \cdot \vec{x}) = f(\sum_j w_j x_j)$$

f is often a simple step function that outputs 1 if its argument is above a threshold and 0 otherwise. However, f may be a more complex function that gives the probability that the observation $\vec{x}$ belongs to a certain class, or outputs a classification based on pseudorandom bits biased according to some distribution determined as a linear function of $\vec{x}$. The coefficients of the linear classifier can come from any source, for example they can result from training a linear machine learning model such as a linear Support Vector Machine (SVM), or be updated in an online way as each new test input is seen.

We observe that there are many reasons why a decision maker may want to protect a policy based on a linear classifier using our techniques: (i.) the decision maker may not wish to disclose which particular algorithm or model it uses to make classifications; (ii.) even if the decision maker is willing to disclose that they use a particular type of model, they may wish to avoid disclosing the trained model weights to protect their intellectual property and to prevent model inversion attacks that would disclose private facts about the underlying training data; (iii.) the decision maker may wish to prevent others with reasonable guesses as to the inputs of a particular subject from learning that subject's classification. (iv.) the decision maker may wish to incorporate random dimensions in its linear policy and may wish for the entire execution to be accountable.

Below, we consider a few concrete linear models and their implementations as circuits suitable for ingestion by our cryptographic tools.

Simple Perceptron

Possibly the simplest example of a linear classifier is the simple perceptron [267], a two-class threshold linear classifier defined as follows on k-dimensional input $\vec{x}$ and trained weights $\vec{w}$.

$$f(x) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Though perceptrons are simple individually, they may be combined into layered networks to build advanced classifiers. Perceptrons form the basis for many modern classification techniques.

In an accountable perceptron, the secret policy $\hat{\mathbf{y}}$ is a k-vector of integers, each subject's private data $\hat{\mathbf{x}}_i$ is a k-vector of booleans, and

$$f(\hat{\mathbf{y}}, \hat{\mathbf{x}}_i) = \left(\sum_{j=0}^{k-1} x_{ij} y_j + b \geq 0 \right)$$

This f is very cheap to represent in an R1CS constraint system. The summation, including the bias term, requires a single constraint, and the comparison to zero requires an additional two constraints. Thus, each perceptron only requires a total of three constraints to represent in R1CS. No randomness is necessary to evaluate this function. Accountable linear models suffer a large overhead because the base models are very compactly representable in R1CS. Indeed, the 1024 constraints of the accountability subcircuit represent a 342× overhead over the three constraints native to a simple linear model. While this overhead is very large, the small absolute time required for the overall circuit mitigates its impact.

We observe that this practice only accounts for the *execution* of the model and not its *training*. While model training presents an interesting accountability issue, we leave that important question open in this work, remarking only that our model allows a decision maker to use any policy it believes will not be ruled unacceptable

on review by the overseer and therefore exactly where any particular secret weight vector originates is beyond the scope of our analysis in this dissertation.

Support Vector Machine

A slightly more complicated linear classifier is a k-dimensional support vector machine (SVM). Originally introduced by Vapnik and Cervonenkis as a special class of perceptron [356], SVMs have become a powerful technique in the machine learning toolbox. SVMs work in the same basic way as perceptrons, but differ in their training approach: an SVM model is the solution to an optimization problem that finds the *maximum margin hyperplane* which separates classes of points (that is, the hyperplane with the largest distance from the nearest training vector; maximizing the margin lowers the generalization error of the classifier, improving its performance on yet-unseen data). Data are mapped into a much higher dimensional space and the perceptron dot product is computed efficiently using *kernels*, or functions that compute a set of dot products with *support vectors*, for which dot products with vectors in the original space are constant. This "kernel trick" allows efficient linear models with many more dimensions than can be achieved with traditional perceptrons.

The model is thus defined (in its dual form) by maximizing the Lagrangian:

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

subject to $\alpha_i \geq 0$ and the constraint that $\sum_{i=1}^{n} \alpha_i y_i = 0$, where the α_i are scalar Lagrange multipliers, the y_i are the dimensions of free vector variables, and the x_i are the training data vectors. $k(x, x')$ is the kernel function described above. Many kernels are possible—one simple kernel is defined by $k(x_i, x_j) = x_i \cdot x_j$, while another more complicated kernel is considered below.

Once the model is trained, an SVM can be represented compactly for evaluation by an arithmetic circuit. SVM evaluation differs from perceptron evaluation only by the additional evaluation of the kernel basis function. The simple inner-product kernel described here requires only a single constraint per classification, leading to a model that requires a total of four constraints per test vector, regardless of dimension, and no randomization. The circuit for such a model is heavily dominated by the cost of the commitment verifications in the accountability subcircuit, which is 1024 constraints. This again represents a large overhead of approximately 257×, also mitigated by the small absolute proving time required for key generation and proving.

RBF-SVM

A more complex kernel-based model is the Radial Basis Function Support Vector Machine (RBF-SVM). A classifier based on such a model returns true for a point x_i if the function

$$f(x_i) = b + \sum_{j=0}^{k} \alpha_j e^{-\gamma \|x_j - x_i\|^2}$$

is non-negative, where b, γ, the α_j's and the x_j's are parameters of the model. This function computes on real numbers, which we can approximate with fixed-point rationals. The most expensive part of the function evaluation is computing the exponential, which can be approximated as follows:

$$e(z) \approx (1 + \frac{z}{n})^n = n^{-n}(n + z)^n$$

for suitably large n. In practice, we can pick n to be 16, and compute the sixteenth power by squaring three times. f can be computed with six constraints per term of the summation: one constraint to compute the magnitude-squared in the exponent, one constraint to multiply by $-\gamma$, three constraints to approximate the exponential, and one to multiply by α_i. Additions require no extra constraints, as they can be

made part of the structure of existing R1CS constraints for free. We therefore require $1024 + 6k$ constraints, where k is the number of support vectors in the kernel. The correct choice of k will depend on the specific model in use. We can guess, however, that k is never more than a few hundred, even in complex models. Therefore, we can conclude that the overall execution time for even a large model will only be a few seconds.

As above, the complexity of the constraint system is dominated by the commitment verifications which require approximately 1024 constraints in total, an overhead of $(1024 + 6k)/6k$ constraints over the total native constraints of the function. This will, in a typical application, be lower than our simpler classifiers, ranging from $1-2\times$ to as little as 15% based on reasonable values of k.

6.3.2 Fair Classification

Chapter 2, Section 2.1.4 describes the growing literature on machine learning classifiers which satisfy formal guarantees of *fairness* either for individuals or for groups. It is natural for us to ask whether such a fairness property can be demonstrated in our system either to the public or to an overseer. Certainly, an overseer who can review the entire decision policy will always be able to determine that such an algorithm was used—our aim is to establish this property without the need for an expensive oversight process. This section sketches the implementation of a very simple fair classifier. Implementation of a more robust fair classifier and testing in the setting of a real classification task are beyond our scope here.

We elect to follow the "fairness through awareness" approach of Dwork et al. [130] and Hardt [202]. This approach requires us to prove that a classifier satisfies the Lipschitz condition

$$D(Mx, My) \leq d(x, y) \qquad \forall x, y \in V$$

That is, the statistical distance D between model outputs Mx and My for test vectors x, y must be bounded by a suitable distance metric $d(x, y)$.

Intuitively, this sort of invariant differs from the types of invariants we have contemplated as extensions to the protocol in Chapter 5, as it is a property not of the computation we wish to hold accountable, but rather of the program and *all possible* data. That is, the property we wish to demonstrate is quantified over the entire model domain V. Naïvely, we could capture such a quantified invariant through a large circuit that considers all possible data points. However, as we will show in this section, we can capture even this type of invariant with an efficient circuit.

We define a simple classifier that is easily proved fair under the Dwork et al. condition but which is likely low in utility.[11] Exploration of classifier utility vs. complexity of arithmetic circuit implementations remains an interesting open problem beyond the scope of this work. Specifically, define a model $M : [0, 1] \longrightarrow \{0, 1\}$ such that $\mathsf{Pr}[M(x) = 1] = x$. That is, the classifier is defined by the random variable $\mathcal{M}(x) = \{1 : x, 0 : 1 - x\}$. Then the total variational distance between classifiers $\mathcal{M}(x)$ and $\mathcal{M}(y)$ is simply:

$$D_{TV}(\mathcal{M}(x), \mathcal{M}(y)) = \frac{1}{2}\left(|x - y| + |(1 - x) - (1 - y)|\right) = \frac{1}{2}\left(|x - y| + |y - x|\right) = d(x, y)$$

Here, D_{TV} is the total variational distance between two distributions and d is the Euclidean norm for real numbers. This proves the fairness criterion for all $x, y \in [0, 1]$.

This classifier can be implemented in a small arithmetic circuit by a technique of multiplying the input by a random number and checking a threshold. We can express this very compactly even if we represent the input and the random numbers in IEEE 794 floating point. Such a representation requires, at the lowest precision, 16 bits per value. We require a single constraint per bit for multiplication and another two constraints per bit for thresholding, using a waterfall approach to designing the

[11] We owe a debt of gratitude to Sean Gerrish for suggesting the outline of this classifier.

circuit. This yields 48 constraints in total; again, the cost of accountable execution is heavily dominated by the cryptographic verification.

6.3.3 Lottery: Diversity Visa Lottery

Each year, the U.S. State Department operates its Electronic Diversity Visa Lottery (DVL) program [353], a program under which non-U.S. residents from countries deemed to have low rates of immigration to the United States may apply for one of 55,000 visas made available annually.

The DVL process operates as follows. Would-be immigrants can apply to be entered in the lottery. Applicants are grouped according to their country of birth. Within each country group, applicants are put into a random rank order list (the lottery step). A (public) number of applicants to accept from each country is calculated, using a formula based on the number of immigrants to the U.S. in recent years from each specific country and region. The calculated number of applicants are selected from the top of each country's rank-ordered list. These "winners" are screened for eligibility to enter the U.S., and if they are eligible they receive visas.

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 lottery 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 lottery initially reported incorrect results, due to programming errors coupled with lax management [159]. In particular, a programming error in the selection process caused the results to be heavily biased towards people who submitted applications in the first two days of the application period.

An accountable implementation of the DVL lottery 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. Further, the cryptographic record generated by the accountable algorithms protocol can help the system's implementers and operators be certain that their policy is being executed as intended, by ensuring that sufficient evidence exists to review every individual decision as well as the set of all decisions.

The key issue is to build a fair k-of-n selection, which is equivalent in complexity to sorting the inputs. Designing small circuits for sorting is a well-studied problem both in theory and in practice. For brevity, we do not summarize any constructions here. However, best-of-breed constructions require $O(k \log n)$ gates. For typical values of $50,000$ selected visa applicants out of approximately 10 million entrants, this yields approximately $825,000$ gates, which require approximately 30 minutes of key generation time and 30 minutes of proving time. Here, the accountability circuit represents an overhead of only 0.12%.

Chapter 7

Conclusion: Designing Computer Systems for Oversight

We conclude this dissertation by remarking on how the tools it presents can be deployed in a real-world context to design automated systems that are amenable to oversight and accountability. Such accountability will only become more important as the decisions being automated become both more consequential and more quotidian. One risk, identified by many authors, is that we accept computerized decision making as inherently opaque, giving up on traditional structures for governance because such systems are too complicated and do not lend themselves to the sort of comprehensibility that affords good governance. We believe the tools introduced in this dissertation provide an effective counterpoint to such disconsolation.

In particular, we argue that our approach which facilitates oversight maps more closely onto real-world accountability requirements than previous works, which have aimed to ensure well defined properties cryptographically but have failed to find real-world adoption. We speculate on this issue in particular, since systems for secure computation have typically been overshadowed by traditional human-driven gover-

nance processes in real deployments, often creating a situation where a system does not meet its security goals as deployed.

Some material in this chapter is drawn from a paper workshopped at the 2015 Privacy Law Scholars Conference which is joint work with Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu [233].

In order for a human process or a traditional bureaucracy to function in an accountable way, accountability must be part of the system's design from the start. This chapter argues that the same is true for computer systems that fulfill important functions. Designers of such systems—and the non-technical stakeholders who often oversee or control system design—must begin with oversight and accountability in mind. Often, computer system designers and detractors of using automated systems to accomplish some task forget that human-driven processes for the same task would also require significant care in order to facilitate oversight and accountability.

Given the ever-widening reach of computer-made decisions, it is essential for computer scientists to know which tools need to be developed and for policymakers to know what technologies are already available. This includes advocating the technologies developed in this dissertation. We offer recommendations for bridging the gap between technologists' desire for specification and the policy process's need for ambiguity. As a first step, we urge policymakers to recognize that accountability is feasible even when an algorithm 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.

7.1 A Remark on Cryptographic Solutions to Real-World Problems

Many cryptographic solutions to real world problems, including to problems of accountability, have been proposed and made the subject of the entire careers of researchers, only to never see any real-world deployment. In this section, we devote some thought to why this general phenomenon exists and our views on why we hope our approach will avoid its pitfalls.

Since the late 1980s, a group of activists referring to themselves as *cypherpunks* advocated the use of strong cryptography to bring about social and political change. They believed cryptography could disintermediate government processes by arbitrating contracts, handling anonymous payments, and providing robust privacy protections while disclosing precisely the information needed in any interaction [210]. A fairly complete vision for a cryptographic society was proposed by Chaum [85]. The cypherpunk movement's rise and impact are chronicled by Levy [246].

Narayanan describes the failure of the cypherpunk dream of a cryptographic utopia to come to fruition as a caution to present-day researchers against the technologically determinist belief that cryptography alone can solve difficult real-world political problems [281]. Narayanan argues that, while cryptography has been very useful in applications that focus on security such as electronic commerce, its success is based on the extent to which the incentives of the entity deploying the cryptography and the parties to the transaction it protects align. In security applications, this alignment is often perfect or nearly so. However, in applications for privacy or which have in mind some kind of social change, these incentives can often be in tension. Specifically, Narayanan argues that the deployment of cryptographic technology reflects the collective will of society.

In a follow-up article, Naryanan addresses the difficulty encountered by protocols from the research literature in seeing real-world deployment [282]. Specifically, Narayanan identifies three major problems in deploying cryptography aimed to improve privacy in a pragmatic (i.e., small, domain-specific) way: (i.) Lack of knowledge among users about the value of cryptographic solutions for privacy, resulting in users not being able to distinguish between functionally equivalent systems with radically different information disclosure properties. (ii.) Human factors concerns, such as the difficulty of key management and the gap between users' observed willingness to disclose data vs. their informed choices about desired disclosures. Human factors also include economic issues such as the business imperative for secondary data uses that would be barred by effective deployment of cryptographic privacy tools. (iii.) The lack of accountability technologies to handle system requirements that do not comport well with typical cryptographic models, such as the "breaking-the-glass" requirement that medical personnel be able to override access control measures on patient records in the event of an emergency. We largely concur with Narayanan's analysis of why most contemporary proposals for cryptographic systems do not make the jump from the research literature to actual practice.

Our methods directly address many of Narayanan's criticisms of cryptographic solutions to social problems. Together with advocacy in communities of practitioners and researchers and the further development of techniques identified in this dissertation, we believe we can address all of the issues raised.

In specific, rather than providing a functionally equivalent replacement with no visible benefit for end users, our protocols extend the functionality of current auditing and oversight processes, enabling them to function more effectively than they currently do while making their operation more transparent to the public. For this reason, our approach does not suffer the problem of many privacy-preserving systems where non-privacy preserving versions seem equally acceptable. Indeed, Chapter 2,

Section 2.2 describes the extensive scholarly literature documenting the insufficiency of current oversight for automated decision processes. Rather, we argue that incentives are properly aligned in the scenarios we consider: a decision maker, its oversight body, and the subjects of its decisions all have an interest in guaranteeing that it is bound to a specific justification for each decision and that this justification complies with social, political, and legal norms for the type of decision at hand.

A related risk from which our protocol does suffer is the risk that the people who could depend on the guarantees our protocol provides will not understand those guarantees or why they can or should depend on them. Further, organizations and government agencies which might deploy our techniques likely do not have the expertise either to understand that such techniques are available (vs. simpler techniques such as simply asserting the correctness of their process, perhaps supported by a program of internal testing and review to increase the decision maker's confidence in such assertions). Nor do potential implementers likely have the skills in-house to develop and deploy such solutions. We can solve this problem through a mix of education and tool development: by teaching key parties, such as rights activists and policymakers, what guarantees are possible, we can create demand for techniques such as ours; by further developing the tools we introduce in this thesis, we can hope to make programming for verified computation systems only marginally different from traditional programming, making it possible for decision making agencies to meet that demand. Of course, it would be excellent if we could call for a broad understanding of the underlying cryptography and the use of cryptography to guarantee the invariants we protect, but we see the path to understanding as shorter when it passes through trusted experts, politically accepted oversight authorities, and socially accepted processes such as public rule makings or adversarial court proceedings. Therefore, while our protocol risks passing decision makers by due to its implementation complexity and esoteric tool chest, we can combat this by educating important figures with the

power to influence decision makers, their oversight bodies, or their constituencies. We can equally combat the difficulty of implementing our techniques by distributing easy-to-use tools. We aim to do both, publishing these techniques in both the computer science and public policy arenas and further developing our tools to enable adoption.

Second, Narayanan suggests that human factors can hinder cryptographic deployment. We remark here that our techniques do not require any additional secrecy—all new cryptographic keys required by general-purpose accountable algorithms are public keys that must be published. Key generation poses a problem, but one which can be surmounted by distributed key generation.[1] Particular applications which have need of randomized algorithms will require trustworthy sources of verifiable randomness, but many of these applications (such as lotteries) already have this requirement.

Finally, Narayanan cites the inflexibility of cryptographic security models with respect to the needs of real application scenarios as a cause for the non-deployability of cryptographic solutions. But indeed, our goal is to bridge precisely this gap by introducing extra flexibility into security models, arguing that oversight and after-the-fact review must be considered, indeed enabled, in order for a protocol to truly provide accountability. Narayanan specifically cites the lack of accountability technologies as a cause of this inflexibility. We believe this dissertation speaks directly to this concern, as well as the related concerns of several authors including Barocas [26], Citron [94, 97], Pasquale [294], and Nissenbaum [153, 287].

[1] It is worth observing that distributed key generation for zk-SNARKs was developed in response to the need to deploy keys for particular SNARK circuits used in an anonymous payments application [265].

7.2 Designing Computer Systems for Procedural Regularity

The first step in any plan to govern an automated system should be to enable the people overseeing it—whether they are government officials, corporate executives, or members of the public—to know how the system makes decisions. A baseline requirement in most contexts is procedural regularity: each participant will know that the same procedure was applied to him and that the procedure was not created in a way that targets individuals.

Specifically and as we have noted in earlier chapters, the tools introduced in this dissertation—especially the protocol of Chapter 5—can be used to demonstrate that:

- The same policy was used to render each decision.
- The policy was fully specified (and this choice was recorded reliably) before the participants were known, removing the ability to design the procedure 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.

A naïve solution to the problem of verifying procedural regularity is to demand transparency of the source code, 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, and transparency has often been suggested as a remedy to accountability issues for automated decision making systems. However, transparency alone

is not sufficient to provide accountability in all cases, as we described in Chapter 2, Section 2.2.3.

Another often suggested approach to the assurance of procedural regularity is to combine transparency with auditing. Auditing treats a decision process as a black box, of which one can see only the inputs and outputs, but not the inner workings. We described calls for such an approach in Chapter 2, Sections 2.2.3 and 2.2.4. 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 their behavior, as we discuss in Chapter 2, Section 2.1.1.

Instead, our approach harnesses the power of computational methods and does not take the design of the computer system as given. Nor do we give up on governance when a component of that system or its design must remain secret. Specifically, we use a suite of cryptographic tools, especially cryptographic commitments and zero-knowledge proofs to guarantee the counterintuitive property that even when a computer program or its input data are secret, the computer system satisfies the requirements for procedural regularity; namely that the same algorithm was used for each decision; that the program implementing that algorithm was determined and recorded before inputs were known; and that outcomes are reproducible. Just because a given policy is secret does not mean that nothing about that policy or the system implementing it can be known.

7.3 Designing Computer Systems for Verification of Desirable Properties

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 often 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.

In the previous section, we described methods for ensuring that automated decisions are reached in accordance with agreed rules, a goal we called procedural regularity. But even when a rule is properly applied in each case, people may still wonder whether the rule operates in aggregate to unfairly disadvantage certain groups—whether, in the policy maker's sense of the term, the rule is discriminatory.

In this section, we turn to the latter question. Beyond ensuring that the rules are consistently applied, how and where can computer science help to illuminate and address undesirable consequences that even a consistently applied rule may have for certain groups? What makes a rule count as unacceptably discriminatory against some person or group is a fundamental, and fundamentally 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.

Non-discrimination is a more complicated goal than procedural regularity, and the solutions that currently exist to address it are less comprehensive. Technical tools offer some ways to ameliorate these problems, but they generally require a well defined notion of what sort of fairness they are supposed to be enforcing. Violations of procedural regularity are clear-cut; violations of principles of fairness are often murky and difficult to define, and thus to demonstrate. However, we remark that without a well established verification of procedural regularity, one can hardly hope to verify any sort of more complex invariant about discrimination or another notion of fairness.

In addition, the precision of computer code often brings into sharp focus the tensions within current legal frameworks for anti-discrimination. Computers favor hard and fast rules over the subjective standards endemic to our common law system in general, and to civil rights law and policy in particular. This suggests that doctrinal reform may be necessary before anti-discrimination law can be satisfactorily applied in this area, as is suggested by Citron and Pasquale [97] and Barocas and Selbst [26]. In Chapter 2, Section 2.1.4, we gave an overview of the concept of fairness in computer science and its many conceptions. More relevantly, though, we gave an overview of the current state of technical tools that speak to anti-discrimination as it is conceived in public policy.

As we demonstrated in Chapter 6, Section 6.3.2, our methods allow us to move beyond procedural regularity and enforce invariants which reflect these state-of-the-art techniques for making fair decisions. When an invariant can be specified ahead of time, more efficient tools can be brought to bear, as exemplified by the protocol and system we present in Chapter 4. We cannot claim to have solved any major societal problem in this dissertation, although we believe that our techniques could be deployed to improve the state of practice in this area and will continue to advocate for such advances. We remark that many important problems remain open, such as more completely integrating our protocols with systems that provide well defined privacy and fairness guarantees. Most importantly, however, computer scientists can work to better sketch the form of fairness properties easily achievable in particular applications, developing robust and efficient machine learning tools that complement our techniques for governing individual decisions. Hardt provides an excellent exemplary study in this area [202].

A significant concern about automated decision making 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 built in. In addition, decisions made by computer systems may enjoy an undeserved aura of fairness or objectivity. However, the design and implementation of computer systems is vulnerable to a variety of problems that can result in systematically faulty and biased determinations, as we summarized in various sections of Chapter 2.

7.4 Fostering Collaboration between Computer Science, Law, and Public Policy

Human-driven decision systems often feel intuitively more trustworthy because we believe humans can exercise *discretion* when applying the rules, seeing when bending or breaking a rule will improve outcomes and acting accordingly. To prevent abuse, such systems often use some degree of *oversight* to enable discretion but detect and ameliorate any harms it causes. Because it is difficult to foresee all situations in which a rule is applied in advance, such flexibility is often considered to be valuable when evaluating the trustworthiness or accountability of some important process. After all, if we could foresee every application of a rule, we could simply bake the "correct" outcome into the rule itself!

Our methods address this gap directly—by focusing on enabling oversight, we allow decision makers the flexibility of discretion when designing a decision policy and may even enable a world in which decisions are processed automatically, but where the policies associated to those decisions are expansive and flexible enough to address concerns with their rigidity and blindness to possible unfairness or sources of discrimination.

Further, by reviewing a decision policy for the places where discretion can be most helpful, a decision maker can focus on what outcomes it desires and whether

its policy achieves those outcomes. Indeed, if individual decision outcomes are to be subject to oversight scrutiny later, decision makers have a strong incentive to review their practices and the outcomes they generate to maximize the likelihood that those practices will be acceptable to the overseer. In turn, if the overseer is responsive to pressures from the public or the law, acceptability to the overseer will imply various degrees of concordance with public values or compliance with the law.

7.4.1 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 a computer system. 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.

However, such a mindset can conflict deeply with many sources of authority to which developers may be responsible. Public opinion and social 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 least intuitively for computer scientists, the operations of U.S. law and policy also work against clear specifications. Below, we explain in more detail why these processes often create ambiguous laws, leaving details—or sometimes even major concepts—open to interpretation.

One cause of this ambiguity is the political reality of legislation. Legislators may be unable to agree on details but able to get votes to pass relatively vague language. Different legislators may support conflicting specific proposals that can be encompassed by a more general bill. Alternatively, legislators might not know precisely what they want but still object to a particular proposed detail; each detail causing

sufficient objections would need to be stripped out of a bill before it could become law.

Another explanation of ambiguity is uncertainty about the situations to which a law or policy will apply. Drafters may worry that they have not fully considered the space of possibilities and may want to build in enough flexibility to cover unexpected circumstances. Incorporating this kind of give can allow a law or policy to grapple with not only current situations but also future technologies or developments that were impossible to anticipate. The U.S. Constitution is often held up as an example of this benefit: generalized provisions for governance and individual rights continue to be applicable even as the landscape of society changes dramatically.

Finally, ambiguity may stem from shared uncertainty about what tactic is best. Here, drafters may feel that they know what situations will arise but still not know how they want to deal with them. Vagueness supports experimentation to help determine what methods are most effective or desirable.

The United States has a long history of dealing with these ambiguities through after-the-fact oversight by the courts. Disagreements about the application of a law or regulation to a specific set of facts can be resolved through cases, and the holes of ambiguity are filled in by the accretion of many rulings on many different, specific situations. Though statutes and regulations may have specific and detailed language, they are expected to be interpreted through caseswith appropriate deference given to the expertise of administrative agencies. Those cases form binding precedents, which in the U.S. common law system, are as authoritative a source of law as the statutes themselves. 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.

The above discussion is intended to inform computer scientists that no one will remove all the ambiguities and offer them a clear, complete specification. Although law- and policymakers can offer clarifications or other changes to guide the work done by developers, drafters may be not only unable to remove certain ambiguities for political reasons but also unwilling to do so because of their desire to imbue the law with flexibility. 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 automated decision making systems.

In practice, these characteristics imply that computer scientists should focus on creating computer systems that are reviewable, not just compliant with the specifications that are generated in the drafting process. It is not enough to build a system in accord with a particular specification—overseers evaluating the decision at a later point in time also need to be able to certify that fact. It would be good for the Diversity Visa Lottery described in Chapter 6, Section 6.3.3 to use a technique for making fair random choices; it would be better for the State Department to be able to demonstrate that property to a court or a skeptical lottery participant.

The technical approaches described in this dissertation provide several ways for computer system designers to ensure that properties of the basis for a decision can be verified later. With these tools, reviewers can check whether a particular program actually was used to make a specific decision, whether random inputs were chosen fairly, and whether that program comports with certain principles specified at the time of the design. Essentially, these technical tools allow continued after-the-fact evaluations of computer systems by allowing for and assisting the judicial system's traditional role in ultimately determining the legality of particular decision making.

Implementing these changes would improve the accountability of automated decision making systems 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.

7.4.2 On Accountable Algorithms and Fuller's "Failures of Law" Argument

We recall briefly the philosophy of law promulgated by Lon Fuller in *The Morality of Law*, which argues for the function of legal systems in terms of a list of *failures*, or desirable properties that a system of laws might fail to provide. Here, we address each of Fuller's failings in turn. Our techniques:

(i.) Show that decisions by an automated system are well justified and followed a set of procedures that were precisely defined at the time of a decision.

(ii.) Allow such portion of that justification as is reasonable in a particular instance to be published, while retaining the capability to keep secret things which ought not be disclosed.

(iii.) Give the opportunity to disclose to the public an understandable version of the rules governing a process. Further, our techniques enable oversight bodies to review these rules for consistency with the rules actually used in specific decision instances. While our techniques do not directly speak to the understandability of a computational process, they do provide a firm ground on which to build such understanding. Together with informed overseers and education of decision subjects, we believe our techniques can make computer systems less opaque, reducing the fears some authors have raised that such systems are "black boxes" that exist beyond the strictures of governance and accountability.

(iv.) Our techniques can prove directly that decision policies predate the decisions in which they are used, either to a decision subject or to a responsible oversight body.

(v.) While our techniques do not prevent contradictory decisions, they do make the justification for decisions available for review. While courts in many countries rely on an intellectual framework of *stare decisis*, our oversight-driven approach allows increased flexibility over traditional *a priori* specification of goals, policies, and invariants in computer systems. This flexibility, in turn, means that the presence or absence of contradictions is determined by inherently political adjudication processes. However, our techniques at least expose the possibility of such political accountability.

(vi.) Again, our framework does not directly address whether automated decision policies are reasonable or proportionate to a particular set of circumstances. However, they do make information available, at least to an overseer, which can facilitate discussion and resolution of this political question.

(vii.) Our techniques make it precisely clear when decision policies have changed, creating incentives for stable decision policies and enabling oversight authorities and the public to understand when the facts of a policy have changed. With traditional natural language policy disclosures, because there is no tight coupling of the policy disclosure to actual decisions, the disparity between disclosed and actual practice can easily grow quite large before new policies are announced.

(viii.) Our techniques precisely address the question of whether and when there is a gap between the specification of a policy and its execution. Simply put, under our protocol, such divergence cannot happen without immediate detection.

In short, our techniques directly remediate many possible failings of systems of laws and rules, while enabling further improvement by informing politically driven oversight.

7.4.3 Recommendations for Law- and Policymakers

The other side of the coin is that law- and policymakers need to recognize and adapt to the changes wrought by automated decision making. Characteristics of computer systems offer both new opportunities and new challenges for the development of legal regimes governing decision making: automated decision making can reduce the benefits of ambiguity, increase accountability to the public, permit accountability even when aspects of the decision process remain secret.

Reduced Benefits of Ambiguity

Although computer scientists can build computer systems 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, which will be applied broadly and at high speed due to the automated nature of the types of decisions we are considering. Furthermore, as we have explained throughout this dissertation, after-the-fact accountability can be limited by algorithmic design—in other words, if an algorithm 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.[2] Both of these traits imply that automated decision making can exacerbate certain disadvantages of legal ambiguities.

In the framework set forth above, we identify three possible explanations for ambiguity: political stalemate, uncertainty about circumstances, and desire for policy experimentation. Here, for each of these cases, we will discuss how the calculation of the relative advantages of precision versus ambiguity shifts when applied to automated decision making and will offer suggestions for retaining the intended benefits of the U.S. lawmaking system.

[2]See our discussion of the problems with transparency and especially the mention of Rice's Theorem in Chapter 2, Section 2.2.3.

Ambiguity stemming from political stalemate essentially passes the buck for determining details from legislators to someone later 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. Currently, judges and bureaucrats ideally would fill this role: courts are expected to offer impartial decisions resistant to public pressure; administrative agencies similarly are expected to retain staff despite changes in political administrations, and those staff members also should offer subject matter expertise beyond what is expected of legislators.

However, this transfer of responsibility sometimes works in less than ideal ways. Automated decision making may exacerbate these problems by adding another actor to whom the responsibility can devolve: the developer who programs the decision making software. Citron offers examples of failures in automated systems that determine benefits eligibility, airport "No Fly" lists, terrorist identifications, and punishment for "dead-beat" parents [94]. Lawmakers should consider this possibility because (i.) the program implementing a particular policy will apply broadly, affecting all participants, (ii.) the developer is unlikely to be held accountable by the current political process, and (iii.) the developer is unlikely to have expertise about the decision being made. All of these factors suggest that lawmakers should avoid giving the responsibility for filling in details of the law to developers. One potential method for restricting the discretion of developers—without requiring specifications in the legislation itself—would be to legislate guidance of software development by administrative agencies. Difficulties in translating between code choices and policy effects still would exist, but could be eased using the technical methods we have described. For example, administrative agencies could work together with developers to identify the properties they want the computer system to possess, and the system then could be designed to permit proof that it satisfies those properties.

Ambiguity generated by uncertainty about circumstances or by a desire for policy experimentation presents a more direct concern. Here, the problem raised by automated decision making is that a computer system locks in a particular choice of the code implementing some policy for the duration of its use. Especially in government contexts, provisions may not be made to update the code. Worries about changing or unexpected circumstances could be assuaged by adding sunset provisions to deployments, requiring periodic review and reconsideration of the software. Computer systems additionally could be designed with eventual revisions and updates in mind. As for preserving the benefits of policy experimentation, the traditional solution might be having multiple algorithms, perhaps implemented by independent teams restricted in their communication; a more sophisticated version of this solution is the incorporation of machine learning into decision making systems. Again, machine learning can have its own fairness pitfalls, and care should be taken to consider fair machine learning methods and to build in precautions like persistent testing of the hypotheses learned by a machine learning model.

More generally, the benefits of ambiguity decrease in the case of automated decision making: the details may be determined by an uninformed actor and then applied broadly and swiftly; the choice of algorithm to implement a particular policy and the implementation of that algorithm in code cements the particular policy choices encoded in a computer system for as long as it is used. Drafters should consider whether they instead should increase the specificity offered by law and policy governing these systems.

To a certain extent, this question mirrors the old *rules versus standards* debate in the legal literature [224, 328] 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"). Rules give clarity and forewarning; standards offer greater flexibility for interpretation.

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 (i.) specifications of each of the properties that they want to ensure that a computer system possesses and requirements that developers design the system in a way that renders those properties provable upon review; alongside (ii.) a general statement of purpose for the computer system. Doing so would give the developer some flexibility in writing the code while also ensuring that particular properties can be checked later.

Accountability to the Public

Oversight is traditionally performed by courts, enforcement agencies or other designated entities such as government prosecutors. 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. The use of computer systems can alter how effectively the legal system and the public can oversee the decision making process.

In one sense, automated decision making 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 presented in this dissertation enable verifications by the public and third parties that are not completely independent from the organizations using our techniques. For example, technologically proficient members of the public or third parties could perform the verifications that a particular policy was used or that it has particular properties. In addition, open-source tools

could be built for participants to check these properties for their own outcomes so that non-technical users could perform these verifications, while the system itself could be overseen by others—potentially both inside and outside of government—who do have the needed technological expertise. As another example, third parties could be involved in generating fair randomness.

Accountability by way of public oversight can also help assuage concerns about data use and collection practices—if members of the public can verify that the results of consequential decisions were well justified, they may be more willing to allow decision makers to collect and retain other data, since they know these data do not bear on important decisions. This, in turn, improves the overall ecosystem of automated decision making because many techniques for guaranteeing formally that decisions are fair require awareness of protected status information, as described in chapter 2.

In contrast to the possibility for enhanced public accountability, the use of computer systems and the reliance on technical tools for oversight can also reduce accountability to the public by hampering the traditional court-based scrutiny of decision making. The U.S. court system is designed to protect against wrongful government actions through the power of judicial review. 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.

However, the use of technical tools shifts the determination of regularity from the courts and enforcement agencies to other parties, specifically external experts or the organizations using the computer systems themselves. This arises because the courts and enforcement agencies are no longer making the determination whether the rules have been properly applied. The determination shifts to the experts evaluating

the automated decision making process. One way to address this unintended shift in responsibility might be 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.

Another issue that challenges public accountability is the validation of technical tools such as those presented in this dissertation. For courts, technical tools cannot be accepted until their integrity and reliability are proven. Courts, though, have long confronted the problem of the admissibility of scientific evidence. For example, the courts took years during the 1980s and 90s to establish and accept the scientific validity of DNA and the methods used to isolate DNA. The Federal Rules of Civil Procedure now provide for the acceptability of new scientific methods in adversarial proceedings. In 1993, the Supreme Court set out standards to meet the Federal Rules requirements that include testing, peer review and publication. This addresses the validation of technical tools used to examine automated decision making, but still leaves 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 automated processes.

Secrets *and* Accountability

Implementing automated decision making in a socially and politically acceptable way requires advances 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 nor always useful as described in Chapter 2, Section 2.2.3. However, disclosing nothing about the basis for a decision is socially unacceptable and generally poses a technical challenge.[3].

[3]Indeed, in 2015, we confront nearly daily news reports of data breaches affecting large companies and governments and their decision making processes

Law- and policymakers should remember that it is possible to make a computer system accountable without the evaluator having full access to the system.

U.S. law and policy often focuses upon transparency or even equates oversight with transparency for the overseer. 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; juvenile records may be sealed because of a decision that youthful mistakes should not follow one forever; and other evidence is deemed inadmissible for a multitude of reasons, including being unscientific, hearsay, inflammatory, or illegally obtained. Thus, precedent exists for basing oversight on partial information.

Strong policy justifications exist for holding back information in the case of automated decision making. Revealing decision policies, the code implementing them, and their input data can expose trade secrets, violate privacy, hamper law enforcement, or lead to gaming of the decision making process. One advantage of computer systems over human-mediated decision processes is that concealment of code and data does not imply an inability to analyze that code and data. The technical tools we describe in this dissertation give law- and policymakers the ability to keep algorithms and their inputs secret while still rendering them accountable. They can apply these tools by implementing them in government-run computer systems, such as the State Department's Diversity Visa Lottery (see Chapter 6, Section 6.3.3), and to provide incentives for non-governmental actors to use them, perhaps by mandating that use or by requiring transparency—at least to designated overseers—of decision policies and their inputs if they do not employ such technical tools.

7.4.4 The Sufficiency of Accountability

Much of our discussion so far has taken accountability as a *necessary* goal for computer systems to achieve trust and acceptance by those who depend on them. Here, we argue that accountability is also *sufficient* to achieve such trust. This question is inherently a political one, since it aims to answer the question of why people trust institutions. Because of this, the question is well studied in political and social theory. We deliberately place a summary of this literature outside the scope of this section, however, as it would detract from our goal of attempting an answer relevant to the material of this chapter and this dissertation.

Accountability as we have defined it implies that the social, legal, and political structures that govern an automated process can function as they are intended to function. While they may not function perfectly and may produce unfair or unjust outcomes in individual cases, those structures exist because they constitute the best compromise for governance available in a society, and one the members of that society are willing to participate in.

Our definition of accountability also requires that the public be able to verify that an automated process satisfies the governance requirements just mentioned. In the most trivial case—that of a process which executes purely deterministically and for which none of the process rules or input are secret—this can be accomplished by full transparency. An audit log that discloses how the process operated can provide the necessary public answerability required to make the process accountable to governance structures if it creates unacceptable outcomes. In more complicated cases, we merely require that the public be able to verify that some trusted authority, vested with the powers of governance, be able to review the process for correctness or compliance with public values and the law. In this way, the answerability of a process for unacceptable behavior devolves to the designated oversight body, such that trust in the automated

process itself relies on trust in the oversight body. This trust, in turn, relies on good governance of the oversight body.

One possible base case for this recursion can be found in social contract theory (summarized briefly in Chapter 2, Section 2.2.1), which argues that the willingness of members of a society to participate in that societies institutions is a proxy for their acceptance of those institutions. The social contract also gives a kind of ultimate answerability for governance of institutions in the form of the withdrawal of consent to participate.

In practical terms, our theory suggests a way forward for the governance of automated decision processes: we wish to further develop the tools presented in this dissertation but also to encourage the agencies within and outside government that oversee the deployment of consequential automated decision systems to adopt either our techniques or rules demanding that the processes they review be constructed in such a way as to facilitate that review. To that end, we advocate for a greater focus on accountability and oversight within the computer science community and simultaneously work to improve the extent of knowledge of available techniques for practitioners and researchers in the law and policy spaces.

Also practically, we remark that review of a process to design an oversight regime forces much important consideration of the specification for that process, since whatever specification it gets, that design will be available for outside review later, even if only to trusted third parties. This provides an incentive for decision makers to attempt to be compliant with social, legal, and political norms *ex ante*, so as not to find themselves facing punishment or loss of face later. It may also present an opportunity for the deployment of the many robust technical measures and formal methods developed to ensure the correspondence of systems as deployed with systems as designed.

We firmly believe that these techniques, properly deployed, can help improve the governance and oversight of algorithms. Many authors have questioned the rush towards automation for consequential processes with a broad impact, whether operated by government or privately ordered [24, 26, 94–97, 106, 121, 138, 153, 287, 293–295, 326, 327]. We are sanguine, however, on the prospects for governable automated processes; automation provides a framework by which to investigate properties of decision making processes which are simply not available for review when the process is executed in the mind of a human.

Bibliography

[1] Hal Abelson, Ross N Anderson, Steven Michael Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G Neumann, Ronald L Rivest, Jeffrey I Schiller, and Bruce Schneier. The risks of key recovery, key escrow, and trusted third-party encryption. *Digital Issues, No. 3*, 1998.

[2] Harold Abelson, Ross Anderson, Steven M Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G Neumann, Ronald L Rivest, Jeffery I Schiller, Bruce Scheier, Michael Specter, and Daniel J Weitzner. Keys under doormats: Mandating insecurity by requiring government access to all data and communications. Technical Report MIT-CSAIL-TR-2015-026, Massachussets Institute of Technology, Computer Science and Artificial Intelligence Laboratory, 2015.

[3] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 674–689. ACM, 2014.

[4] Nabil R Adam and John C Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys (CSUR)*, 21(4):515–556, 1989.

[5] Ben Adida. *Advances in cryptographic voting systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[6] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.

[7] Ben Adida and Ronald L Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 29–40. ACM, 2006.

[8] Alfred V. Aho and John E. Hopcroft. *Design & Analysis of Computer Algorithms*. Addison Wesley, 1974.

[9] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108. ACM, 1996.

[10] David G Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable internet protocol (aip). In *ACM SIGCOMM Computer Communication Review*, volume 38:4, pages 339–350. ACM, 2008.

[11] Ross Anderson. *Security engineering*. John Wiley & Sons, 2008.

[12] Andrew W Appel. *Program logics for certified compilers*. Cambridge University Press, 2014.

[13] K Argyraki, P Maniatis, O Irzak, and S Shenker. An accountability interface for the internet. In *Proc. 14th ICNP*, 2007.

[14] Aristotle. *Politics, A Treatise on Government*. J.M. Dent Sons, London, 1928. trans. William Ellis, A. M.

[15] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[16] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; a new characterization of NP. In *Proc. FOCS*, 1992.

[17] Charles Asmuth and John Bloom. A modular approach to key safeguarding. *IEEE transactions on information theory*, 29(2):208–210, 1983.

[18] Robert J Aumann and Michael Maschler. Game theoretic analysis of a bankruptcy problem from the talmud. *Journal of economic Theory*, 36(2):195–213, 1985.

[19] AK Austin. Sharing a cake. *The Mathematical Gazette*, pages 212–215, 1982.

[20] Michael Backes, Anupam Datta, Ante Derek, John C Mitchell, and Mathieu Turuani. Compositional analysis of contract signing protocols. In *Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop*, pages 94–110. IEEE, 2005.

[21] Michael Backes, Peter Druschel, Andreas Haeberlen, and Dominique Unruh. CSAR: a practical and provable technique to make randomized systems accountable. *Proc. NDSS*, 2009.

[22] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[23] Josep Balasch, Alfredo Rial, Carmela Troncoso, Bart Preneel, Ingrid Verbauwhede, and Christophe Geuens. Pretp: Privacy-preserving electronic toll pricing. In *USENIX Security Symposium*, pages 63–78, 2010.

[24] Kenneth A Bamberger. Technologies of compliance: Risk and regulation in a digital age. *Tex. L. Rev.*, 88:669, 2009.

[25] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in cryptologyCRYPTO 2001*, pages 1–18. Springer, 2001.

[26] Solon Barocas and Andrew D. Selbst. Big data's disparate impact. *Available at SSRN 2477899*, 2014. `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2477899`.

[27] Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[28] Adam Barth, Anupam Datta, John C Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[29] Adam Barth, John C Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *Computer Security Foundations Symposium, 2007. CSF'07. 20th IEEE*, pages 279–294. IEEE, 2007.

[30] Adam Bates, Kevin Butler, Micah Sherr, Clay Shields, Patrick Traynor, and Dan Wallach. Accountable wiretapping–or–i know they can hear you now. *Journal of Computer Security*, 2015.

[31] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[32] Amos Beimel. Secret-sharing schemes: a survey. In *Coding and cryptology*, pages 11–46. Springer, 2011.

[33] Giampaolo Bella and Lawrence C Paulson. Accountability protocols: Formalized and verified. *ACM Transactions on Information and System Security (TISSEC)*, 9(2):138–161, 2006.

[34] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*. ACM, 1997.

[35] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 57–66. ACM, 1995.

[36] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *Advances in CryptologyEurocrypt96*, pages 399–416. Springer, 1996.

[37] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. *Proc. CCS*, 2008.

[38] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete-efficiency threshold of probabilistically-checkable proofs. Technical report, Electronic Colloquium on Computational Complexity, 2012. `http://eccc.hpi-web.de/report/2012/045/`.

[39] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. *CRYPTO*, 2013.

[40] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *IEEE Symposium on Security and Privacy*, 2015.

[41] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von Neumann architecture. Technical report, Cryptology ePrint Archive, http://eprint.iacr.org/2013/879, 2013.

[42] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security*, pages 781–796, 2014.

[43] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Advances in CryptologyCRYPTO86*, pages 251–260. Springer, 1987.

[44] Guido Bertoni, Joan Daemen, Michaël Peeters, and GV Assche. The keccak reference. `http://keccak.noekeon.org`, 2011.

[45] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3:30, 2009.

[46] Big Data: seizing opportunities, preserving values. `http://www.whitehouse.gov/issues/technology/big-data-review`, May 2014.

[47] Tom Bingham. *The rule of law*. Penguin UK, 2011.

[48] Matt Bishop. A standard audit trail format. In *Proceedings of the 18th National Information Systems Security Conference*, pages 136–145, 1995.

[49] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[50] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography*, pages 315–333. Springer, 2013.

[51] George Robert Blakley. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, volume 48, pages 313–317, 1979.

[52] Matt Blaze. Protocol failure in the escrowed encryption standard. In *CCS*, pages 59–67. ACM, 1994.

[53] Jeremiah Blocki, Nicolas Christin, Anupam Datta, Ariel D Procaccia, and Arunesh Sinha. Audit games. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 41–47. AAAI Press, 2013.

[54] Manuel Blum. Coin flipping by telephone: A protocol for solving impossible problems. *CRYPTO*, 1981.

[55] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.

[56] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. *Proc. STOC*, 1988.

[57] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. *Financial Cryptography*, 2009.

[58] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public key cryptographyPKC 2003*, pages 31–46. Springer, 2002.

[59] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT 2004*, pages 223–238. Springer, 2004.

[60] Dan Boneh and Xavier Boyen. Efficient selective identity-based encryption without random oracles. *J. Cryptology*, 24(4):659–693, 2011.

[61] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.

[62] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*, pages 506–522. Springer, 2004.

[63] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *CRYPTO 2001*, pages 213–229. Springer, 2001.

[64] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[65] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273. Springer, 2011.

[66] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. *Proceedings of IEEE Security and Privacy*, 2015.

[67] Colin Boyd. Digital multisignatures. In *Cryptography and Coding*. Institute of Mathematics and its Applications, Clarendon Press, 1986.

[68] Steven J Brams and Alan D Taylor. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

[69] Steven J Brams and Alan D Taylor. *The win-win solution: guaranteeing fair shares to everybody*. WW Norton & Company, 2000.

[70] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[71] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *CRYPTO86*, o1986.

[72] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 341–357. ACM, 2013.

[73] Sonja Buchegger and Jean-Yves Le Boudec. A robust reputation system for mobile ad-hoc networks. Technical Report LCA-REPORT-2003-023, École Polytechnique Fédérale de Lausanne, Laboratory for Communications and Applications, 2003.

[74] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *Database TheoryICDT 2001*, pages 316–330. Springer, 2001.

[75] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks*, pages 141–155. Springer, 2006.

[76] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in CryptologyEUROCRYPT 2001*, pages 93–118. Springer, 2001.

[77] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in CryptologyCRYPTO 2002*, pages 61–76. Springer, 2002.

[78] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT*, 2007.

[79] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[80] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.

[81] Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K Reiter, Ronitt Rubinfeld, and Rebecca N Wright. Selective private function evaluation with applications to private statistics. *Proc. PODC*, 2001.

[82] John Canny and Stephen Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology-EUROCRYPT 2004*, pages 138–152. Springer, 2004.

[83] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology—CRYPTO'82*, pages 199–203. Springer, 1982.

[84] David Chaum. Blind signature system. In *Advances in cryptology—CRYPTO'83*, pages 153–153. Springer, 1983.

[85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[86] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. *Proc. STOC*, 1988.

[87] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990.

[88] David Chaum and Eugène Van Heyst. Group signatures. In *Advances in CryptologyEUROCRYPT91*, pages 257–265. Springer, 1991.

[89] Liqun Chen, Zhaohui Cheng, and Nigel P Smart. Identity-based key agreement protocols from pairings. *International Journal of Information Security*, 6(4):213–241, 2007.

[90] Francis Y Chin and Gultekin Ozsoyoglu. Auditing and inference control in statistical databases. *IEEE Transactions on Software Engineering*, pages 574–582, 1982.

[91] Seung Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. *Theory of Cryptography*, pages 39–53, 2012.

[92] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985.

[93] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology–CRYPTO 2010*, pages 483–501. Springer, 2010.

[94] Danielle Citron. Technological due process. *Washington University Law Review*, 85:1249–1313, 2007.

[95] Danielle Keats Citron. Open code governance. In *University of Chicago Legal Forum*, pages 355–387, 2008.

[96] Danielle Keats Citron and Frank A Pasquale. Network accountability for the domestic intelligence apparatus. *Hastings Law Journal*, 62:1441, 2011.

[97] Danielle Keats Citron and Frank A. Pasquale. The scored society: Due process for automated predictions. *Washington Law Review*, 89, 2014.

[98] Jeremy Clark and Aleksander Essex. Commitcoin: carbon dating commitments with bitcoin. In *Financial Cryptography and Data Security*, pages 390–398. Springer, 2012.

[99] William Douglas Clinger. *Foundations of actor semantics*. PhD thesis, Massachusetts Institute of Technology, 1981.

[100] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and coding*, pages 360–363. Springer, 2001.

[101] Janet L Colbert and Paul Bowen. A comparison of internal control: Cobit, sac, coso and sas. *IS Audit and Control Journal*, pages 26–35, 1996.

[102] Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, second edition, 2001.

[103] Henry Corrigan-Gibbs, Wendy Mu, Dan Boneh, and Bryan Ford. Ensuring high-quality randomness in cryptographic key generation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 685–696. ACM, 2013.

[104] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *Proceedings of the 36th IEEE Symposium on Security and Privacy, S&P*, volume 15, 2014.

[105] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[106] Kate Crawford and Jason Schultz. Big data and due process: Toward a framework to redress predictive privacy harms. *BCL Rev.*, 55:93, 2014.

[107] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *Advances in CryptologyCRYPTO87*, pages 350–354. Springer, 1988.

[108] RA Croft and SP Harris. Public-key cryptography and re-usable shared secrets. In *Cryptography and Coding*, pages 189–201. Institute of Mathematics and its Applications, 1989.

[109] Scott A Crosby and Dan S Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium*, pages 317–334, 2009.

[110] Bart Custers, Toon Calders, Bart Schermer, and Tal Zarsky. *Discrimination and privacy in the information society: Data mining and profiling in large databases*, volume 3. Springer Science & Business Media, 2012.

[111] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136. Springer, 2001.

[112] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination. *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2015.

[113] Anupam Datta. Privacy through accountability: A computer science perspective. In *Distributed Computing and Internet Technology*, pages 43–49. Springer, 2014.

[114] Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma, and Arunesh Sinha. Program actions as actual causes: A building block for accountability. *Proceedings of the Computer Security Foundations Symposium*, 2015.

[115] Emiliano De Cristofaro, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Privacy-preserving policy-based information transfer. In *Privacy enhancing technologies*. Springer, 2009.

[116] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby–a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security, NDSS*, 2015.

[117] Dorothy E Denning. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, pages 222–232, 1987.

[118] Dorothy E Denning and Dennis K Branstad. Key escrow encryption systems. *Commun. ACM*, 39(3):35, 1996.

[119] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *Advances in CryptologyCrypto87*, pages 120–127. Springer, 1987.

[120] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in CryptologyCRYPTO89 Proceedings*, pages 307–315. Springer, 1989.

[121] Nicholas Diakopoulos. Algorithmic accountability: Journalistic investigation of computational power structures. *Digital Journalism*, 3(3):398–415, 2015.

[122] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[123] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[124] Edsger Wybe Dijkstra. *A discipline of programming*. prentice-hall Englewood Cliffs, 1976.

[125] Yan Zong Ding, Danny Harnik, Alon Rosen, and Ronen Shaltiel. Constant-round oblivious transfer in the bounded storage model. In *Theory of Cryptography*, pages 446–472. Springer, 2004.

[126] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210. ACM, 2003.

[127] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography-PKC 2005*, pages 416–431. Springer, 2005.

[128] Cynthia Dwork. Differential privacy. *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1–12, 2006.

[129] Cynthia Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer, 2011.

[130] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 214–226. ACM, 2012.

[131] Cynthia Dwork and Deirdre K Mulligan. It's not privacy, and it's not fair. *Stanford Law Review Online*, 66:35, 2013.

[132] Ronald Dworkin. *Law's empire*. Harvard University Press, 1986.

[133] Ronald M Dworkin. *The philosophy of law*, volume 102. Oxford University Press Oxford, 1977.

[134] Carl Ellison. Ceremony design and analysis. *IACR eprint archive*, 399, 2007. `http://eprint.iacr.org/2007/399.pdf`.

[135] Steven Englehardt, Christian Eubank, Peter Zimmerman, Dillon Reisman, and Arvind Narayanan. Web privacy measurement: Scientific principles, engineering platform, and new results. *Manuscript posted at http://randomwalker.info/publications/WebPrivacyMeasurement.pdf*, 2014.

[136] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away:

The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299. International World Wide Web Conferences Steering Committee, 2015.

[137] Michael D Ernst, René Just, Suzanne Millstein, Werner Dietl, Stuart Pernsteiner, Franziska Roesner, Karl Koscher, Paulo Barros Barros, Ravi Bhoraskar, Seungyeop Han, et al. Collaborative verification of information flow for a high-assurance app store. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1092–1104. ACM, 2014.

[138] Virginia Eubanks. *Digital dead end: Fighting for social justice in the information age*. MIT Press, 2011.

[139] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[140] John Samuel Ezell. *Fortune's merry wheel*. Harvard University Press Cambridge, 1960.

[141] Csilla Farkas, Gábor Ziegler, Attila Meretei, and András Lörincz. Anonymity and accountability in self-organizing electronic communities. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 81–90. ACM, 2002.

[142] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Proc. FOCS*. IEEE, 1990.

[143] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[144] Joan Feigenbaum. How useful are formal models of accountability? In *2nd International Workshop on Accountability: Science, Technology, and Policy*, 2014.

[145] Joan Feigenbaum, Aaron D Jaggard, and Rebecca N Wright. Towards a formal model of accountability. In *Proceedings of the 2011 workshop on New security paradigms workshop*, pages 45–56. ACM, 2011.

[146] Joan Feigenbaum, Aaron D Jaggard, Rebecca N Wright, and Hongda Xiao. Systematizing accountability in computer science (version of feb. 17, 2012). Technical Report YALEU/DCS/TR-1452, Yale University, New Haven, CT, 2012.

[147] Andre Ferreira, Ricardo Cruz-Correia, Luiz Henrique M Antunes, Pedro Farinha, E Oliveira-Palhares, David W Chadwick, and Altamiro Costa-Pereira. How to break access control in a controlled manner. In *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on*, pages 847–854. IEEE, 2006.

[148] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. *CRYPTO*, 1986.

[149] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 844–855. ACM, 2014.

[150] Benjamin Fish, Jeremy Kun, and Adám D Lelkes. Fair boosting: a case study. *Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2015. `http://www.fatml.org/papers/Fish_Kun_Lelkes.pdf`.

[151] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography*, pages 90–104. Springer, 2001.

[152] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Proceedings of USENIX Security*, 2014.

[153] Batya Friedman and Helen Nissenbaum. Bias in computer systems. *ACM Transactions on Information Systems (TOIS)*, 14(3):330–347, 1996.

[154] Batya Friedman, John C Thomas, Jonathan Grudin, Clifford Nass, Helen Nissenbaum, Mark Schlager, and Ben Shneiderman. Trust me, i'm accountable: trust and accountability online. In *CHI'99 Extended Abstracts on Human Factors in Computing Systems*, pages 79–80. ACM, 1999.

[155] Kazuto Fukuchi, Jun Sakuma, and Toshihiro Kamishima. Prediction with model-based neutrality. In *Machine Learning and Knowledge Discovery in Databases*, pages 499–514. Springer, 2013.

[156] Lon L Fuller. *The morality of law*, volume 152. Yale University Press, 1977.

[157] Simson Garfinkel. *PGP: pretty good privacy.* " O'Reilly Media, Inc.", 1995.

[158] Shelly Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[159] Harold W. Geisel. Memorandum report: Review of the fy2012 diversity visa program selection process. `http://oig.state.gov/system/files/176330.pdf`.

[160] David Gelperin and Bill Hetzel. The growth of software testing. *Communications of the ACM*, 31(6):687–695, 1988.

[161] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[162] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. *EUROCRYPT*, 2013.

[163] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[164] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.

[165] Patrice Godefroid, Michael Y Levin, and David A Molnar. Automated whitebox fuzz testing. In *NDSS*, volume 8, pages 151–166, 2008.

[166] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.

[167] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A. Kroll, Edward W. Felten, and Arvind Narayanan. Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme. *[Online]. http://www.cs.princeton.edu/~stevenag/threshold_sigs.pdf*, 2015.

[168] Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[169] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(42):236–241, 1996.

[170] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[171] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[172] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564. ACM, 2013.

[173] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Advances in Cryptology–CRYPTO 2013*, pages 536–553. Springer, 2013.

[174] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[175] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304. ACM, 1985.

[176] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[177] Shafi Goldwasser, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with an honest majority. *Proc. STOC*, 87:218–229, 1987.

[178] Don Gotterbarn, Keith Miller, and Simon Rogerson. Software engineering code of ethics. *Communications of the ACM*, 40(11):110–118, 1997.

[179] Don Gotterbarn, Keith Miller, and Simon Rogerson. Software engineering code of ethics is approved. *Communications of the ACM*, 42(10):102–107, 1999.

[180] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. *EUROCRYPT*, 2008.

[181] Ruth W Grant and Robert O Keohane. Accountability and abuses of power in world politics. *American political science review*, 99(01):29–43, 2005.

[182] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *ASIACRYPT*, pages 265–282, 2007.

[183] Matthew Green and Susan Hohenberger. Universally composable adaptive oblivious transfer. In *Advances in Cryptology-ASIACRYPT 2008*. Springer, 2008.

[184] Albert G. Greenberg and Neal Madras. How fair is fair queuing. *Journal of the ACM (JACM)*, 39(3):568–598, 1992.

[185] Dimitris A Gritzalis. *Secure electronic voting*, volume 7. Springer Science & Business Media, 2012.

[186] Jens Groth. Short non-interactive zero-knowledge proofs. In *Advances in Cryptology-ASIACRYPT 2010*, pages 341–358. Springer, 2010.

[187] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology–EUROCRYPT 2008*, pages 415–432. Springer, 2008.

[188] Le Guan, Jingqiang Lin, Bo Luo, Jiwu Jing, and Jing Wang. Protecting private keys against memory disclosure attacks using hardware transactional memory. *Proc. IEEE S & P*, 2015.

[189] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical privacy in online advertising. In *NSDI*, 2011.

[190] B Guido, D Joan, P Michaël, and VA Gilles. Cryptographic sponge functions. `http://sponge.noekeon.org/`, 2011.

[191] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu. Asax: Software architecture and rule-based language for universal audit trail analysis. In *Computer SecurityESORICS 92*, pages 435–450. Springer, 1992.

[192] Andreas Haeberlen. A case for the accountable cloud. *ACM SIGOPS Operating Systems Review*, 44(2):52–57, 2010.

[193] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In *OSDI*, pages 119–134, 2010.

[194] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. In *ACM SIGOPS Operating Systems Review*, volume 41:6, pages 175–188. ACM, 2007.

[195] Sara Hajian. *Simultaneous discrimination prevention and privacy protection in data publishing and mining*. PhD thesis, Universitat Rovira i Virgili, 2013.

[196] Sara Hajian, Josep Domingo-Ferrer, and Oriol Farràs. Generalization-based privacy preservation and discrimination prevention in data publishing and mining. *Data Mining and Knowledge Discovery*, 28(5-6):1158–1188, 2014.

[197] Sara Hajian, Josep Domingo-Ferrer, Anna Monreale, Dino Pedreschi, and Fosca Giannotti. Discrimination-and privacy-aware patterns. *Data Mining and Knowledge Discovery*, pages 1–50, 2014.

[198] Sara Hajian, Anna Monreale, Dino Pedreschi, Josep Domingo-Ferrer, and Fosca Giannotti. Fair pattern discovery. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 113–120. ACM, 2014.

[199] Joseph Lorenzo Hall. Election auditing bibliography, 2010. `https://josephhall.org/papers/auditing_biblio.pdf`.

[200] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part i: Causes. *The British journal for the philosophy of science*, 56(4):843–887, 2005.

[201] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part ii: Explanations. *The British Journal for the Philosophy of Science*, 56(4):889–911, 2005.

[202] Moritz AW Hardt. *A study of privacy and fairness in sensitive data analysis*. PhD thesis, Princeton University, 2011.

[203] Herbert Lionel Adolphus Hart. *The concept of law*. Oxford University Press, 1961 (2012).

[204] Paul Helman and Gunar Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *Software Engineering, IEEE Transactions on*, 19(9):886–901, 1993.

[205] Ryan Henry and Ian Goldberg. Formalizing anonymous blacklisting systems. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 81–95. IEEE, 2011.

[206] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[207] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[208] Thomas Hobbes and Edwin Curley. *Leviathan: with selected variants from the Latin edition of 1668*, volume 8348. Hackett Publishing, 1651 (1994).

[209] Russell Housley, Warwick Ford, William Polk, and David Solo. Internet x. 509 public key infrastructure certificate and crl profile. RFC 2459 (Proposed Standard), January 1999.

[210] Eric Hughes. A cypherpunk's manifesto. `http://www.activism.net/cypherpunk/manifesto.html`, 1993.

[211] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Theory of Computing and Systems, 1997., Proceedings of the Fifth Israeli Symposium on*, pages 174–183. IEEE, 1997.

[212] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. Towards a theory of accountability and audit. In *Computer Security–ESORICS 2009*, pages 152–167. Springer, 2009.

[213] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, pages 577–594, 2009.

[214] Stanisław Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. *EUROCRYPT*, 2007.

[215] Deborah G Johnson. Computer ethics. In Luciano Floridi, editor, *the Philosophy of Computing and Information*, pages 65–75. Wiley, 2003.

[216] Jakob Jonsson and Burt Kaliski. Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1. *White Paper*, 2003.

[217] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[218] Ka-Ping Yee. *Building Reliable Voting Machine Software*. PhD thesis, University of California, 2007. `http://zesty.ca/pubs/yee-phd.pdf`.

[219] Rajashekar Kailar. Accountability in electronic commerce protocols. *Software Engineering, IEEE Transactions on*, 22(5):313–328, 1996.

[220] Seny Kamara. Are compliance and privacy always at odds? Available at `http:/outsourcedbits.org//2013/07/23/are-compliance-and-privacy-always-at-odds/`.

[221] Seny Kamara. Restructuring the NSA metadata program. `http://research.microsoft.com/en-us/um/people/senyk/pubs/metacrypt.pdf`, 2013.

[222] Toshihiro Kamishima, Shotaro Akaho, Hidek Asoh, and Jun Sakuma. Considerations on fairness-aware data mining. In *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*, pages 378–385. IEEE, 2012.

[223] Toshihiro Kamishima, Shotaro Akaho, and Jun Sakuma. Fairness-aware learning through regularization approach. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 643–650. IEEE, 2011.

[224] Louis Kaplow. Rules versus standards: An economic analysis. *Duke Law Journal*, pages 557–629, 1992.

[225] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. Simulatable auditing. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 118–127. ACM, 2005.

[226] Orin S Kerr. The fourth amendment and new technologies: constitutional myths and the case for caution. *Michigan Law Review*, pages 801–888, 2004.

[227] Orin S Kerr. A user's guide to the stored communications act, and a legislator's guide to amending it. *George Washington Law Review*, 2004.

[228] Orin S Kerr. Applying the fourth amendment to the internet: A general approach. *Stanford Law Review*, 2009.

[229] Donald Ervin Knuth. *The art of computer programming: fundamental algorithms*, volume 1. Pearson Education, 1968.

[230] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. *Automata, Languages and Programming*, pages 486–498, 2008.

[231] Kreuter, B. and shelat, a. and Shen, C.H. Billion-gate secure computation with malicious adversaries. *USENIX Security*, 2012.

[232] J Kroll, E Felten, and Dan Boneh. Secure protocols for accountable warrant execution. *See http://www. cs. princeton. edu/felten/warrant-paper.pdf*, 2014.

[233] Joshua Kroll, Joanna Huey, Solon Barocas, Edward W. Felten, Joel R. Reidenberg, David G. Robinson, and Harlan Yu. Designing algorithmic decision processes for governance and oversight. *Privacy Law Scholars Conference*, 2015.

[234] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 30–41. ACM, 2014.

[235] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 526–535. ACM, 2010.

[236] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report Technical Report CSL-98, SRI International Palo Alto, 1979.

[237] Butler Lampson. Privacy and security usable security: how to get it. *Communications of the ACM*, 52(11):25–27, 2009.

[238] Butler W Lampson. Computer security in the real world. *Computer*, 37(6):37–46, 2004.

[239] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical Report RFC 6962, Internet Engineering Task Force, 2013.

[240] Geoffrey A Lee. The coming of age of double entry: the giovanni farolfi ledger of 1299-1300. *The Accounting Historians Journal*, pages 79–95, 1977.

[241] Arjen K Lenstra, Peter Winkler, and Yacov Yacobi. A key escrow system with warrant bounds. In *Advances in Cryptology—CRYPT0'95*. Springer, 1995.

[242] Lawrence Lessig. *Code and other laws of cyberspace*. Basic books, 1999.

[243] Lawrence Lessig. *Code: Version 2.0*. Lawrence Lessig, 2006.

[244] Laura M Leventhal, Keith E Instone, and David W Chilson. Another view of computer science ethics: patterns of responses among computer scientists. *Journal of Systems and Software*, 17(1):49–60, 1992.

[245] Nancy G Leveson and Clark S Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[246] Steven Levy. *Crypto: How the Code Rebels Beat the Government–Saving Privacy in the Digital Age*. Penguin, 2001.

[247] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Theory of Cryptography*, pages 329–346, 2011.

[248] Jacques-Louis Lions et al. Ariane 5 flight 501 failure. Technical report, European Space Agency, 1996.

[249] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography*, pages 169–189. Springer, 2012.

[250] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Advances in Cryptology-ASIACRYPT 2013*, pages 41–60. Springer, 2013.

[251] Jia Liu, Mark D Ryan, and Liqun Chen. Balancing societal security and individual privacy: Accountable escrow system. In *27th IEEE Computer Security Foundations Symposium (CSF)*, 2014.

[252] John Locke and Crawford Brough Macpherson. *Second treatise of government*. Hackett Publishing, 1689 (1980).

[253] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

[254] Teresa F Lunt. Automated audit trail analysis and intrusion detection: A survey. In *In Proceedings of the 11th National Computer Security Conference*. Citeseer, 1988.

[255] Teresa F Lunt, R Jagannathan, Rosanna Lee, Alan Whitehurst, and Sherry Listgarten. Knowledge-based intrusion detection. In *AI Systems in Government Conference, 1989., Proceedings of the Annual*, pages 102–107. IEEE, 1989.

[256] Ben Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, 2009.

[257] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay: a secure two-party computation system. *USENIX Security*, 2004.

[258] Marbury v. madison, 1803.

[259] Sarah Meiklejohn, Keaton Mowery, Stephen Checkoway, and Hovav Shacham. The phantom tollbooth: Privacy-preserving electronic toll collection in the presence of driver collusion. In *USENIX Security Symposium*, volume 201:1, 2011.

[260] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. Coniks: Bringing key transparency to end users. *USENIX Security*, 2015.

[261] Wei Meng, Xinyu Xing, Anmol Sheth, Udi Weinsberg, and Wenke Lee. Your online interests: Pwned! a pollution attack against targeted advertising. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 129–140. ACM, 2014.

[262] Ralph C. Merkle. A digital signature based on a conventional encryption function. *CRYPTO*, 1987.

[263] Ralph C Merkle. A certified digital signature. In *Advances in CryptologyCRYPTO89 Proceedings*, pages 218–238. Springer, 1990.

[264] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 245–254. ACM, 2001.

[265] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. *Proc. IEEE S & P*, 2013.

[266] Maurice Mignotte. How to share a secret. In *Cryptography*, pages 371–375. Springer, 1983.

[267] Marvin Minsky and Papert Seymour. *Perceptrons.* MIT press, 1969.

[268] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez-Salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, et al. The provenance of electronic data. *Communications of the ACM*, 51(4):52–58, 2008.

[269] Richard G Mulgan. *Holding power to account: accountability in modern democracies*. Palgrave Macmillan, 2003.

[270] Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo I Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference, General Track*, pages 43–56, 2006.

[271] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo I Seltzer. Provenance for the cloud. In *FAST*, volume 10, pages 15–14, 2010.

[272] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011 (1979).

[273] Shubha U Nabar, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. A survey of query auditing techniques for data privacy. In *Privacy-Preserving Data Mining*, pages 415–431. Springer, 2008.

[274] Shubha U Nabar, Bhaskara Marthi, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. Towards robustness in query auditing. In *Proceedings of the 32nd international conference on Very large data bases*, pages 151–162, 2006.

[275] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `http://bitcoin.org/bitcoin.pdf`, 2008.

[276] Moni Naor. Bit commitment using pseudorandomness. *Journal of cryptology*, 4(2):151–158, 1991.

[277] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254. ACM, 1999.

[278] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *Advances in CryptologyCRYPTO99*, pages 573–590. Springer, 1999.

[279] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *CRYPTO*, pages 573–590, 1999.

[280] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[281] Arvind Narayanan. What happened to the crypto dream?, part 1. *IEEE Security & Privacy*, pages 75–76, 2013.

[282] Arvind Narayanan. What happened to the crypto dream?, part 2. *IEEE Security & Privacy*, pages 68–71, 2013.

[283] Arvind Narayanan and Edward W. Felten. No silver bullet: De-identification still doesn't work. *[Online.]*, 9 July 2014. `http://randomwalker.info/publications/no-silver-bullet-de-identification.pdf`.

[284] Arvind Narayanan, Joanna Huey, and Edward W Felten. A precautionary approach to big data privacy. *Computers, Privacy, and Data Protection*, 2015.

[285] Arvind Narayanan and Shannon Vallor. Why software engineering courses should include ethics coverage. *Communications of the ACM*, 57(3):23–25, 2014.

[286] National Institute of Standards and Technology. FIPS publication 185: Escrowed encryption standard. U.S. Department of Commerce, 1994.

[287] Helen Nissenbaum. Accountability in a computerized society. *Science and engineering ethics*, 2(1):25–42, 1996.

[288] Wakaha Ogata and Kaoru Kurosawa. Oblivious keyword search. *Journal of complexity*, 20(2):356–371, 2004.

[289] Paul Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review*, 57:1701, 2010.

[290] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptologyEUROCRYPT99*, pages 223–238. Springer, 1999.

[291] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Proc. IEEE S & P*, 2013.

[292] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography*, pages 422–439. Springer, 2012.

[293] Frank Pasquale. Reputation regulation: Disclosure and the challenge of clandestinely commensurating computing. In Saul Levmore and Martha C. Nussbaum, editors, *The Offensive Internet: Privacy, Speech and Reputation*. Harvard University Press, 2010.

[294] Frank Pasquale. *The Black Box Society*. Harvard University Press, Cambridge, MA, 2015.

[295] Frank A Pasquale. Trusting (and verifying) online intermediaries' policing. In *The Next Digital Decade: Essays on the Future of the Internet*, page 347. TechFreedom, 2010. Freely available online: `http://nextdigitaldecade.com/ndd_book.pdf`.

[296] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in CryptologyCRYPTO91*, pages 129–140. Springer, 1991.

[297] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. Defy: A deniable, encrypted file system for log-structured storage. *Proc. NDSS*, 2015.

[298] Benjamin C Pierce. *Types and programming languages*. MIT press, Cambridge, MA, 2002.

[299] Benjamin C Pierce. *Advanced topics in types and programming languages*. MIT press, 2005.

[300] Benny Pinkas, Thomas Schneider, Nigel Smart, and Stephen Williams. Secure two-party computation is practical. *ASIACRYPT*, 2009.

[301] Raluca A Popa, Hari Balakrishnan, and Andrew J Blumberg. Vpriv: Protecting privacy in location-based vehicular services. In *USENIX Security Symposium*, pages 335–350, 2009.

[302] John A Price. Gambling in traditional asia. *Anthropologica*, pages 157–180, 1972.

[303] Michael O Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[304] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981. Re-released as Cryptology Eprint Archive Report 2005/187, `https://eprint.iacr.org/2005/187`.

[305] Michael O Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

[306] John Rawls. *A theory of justice*. Harvard university press, 1971 (2009).

[307] Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, and Neil Stratford. Xenoservers: Accountable execution of untrusted programs. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 136–141. IEEE, 1999.

[308] ACM Committee on Computers and moderator Public Policy, Peter G. Neumann. `http://catless.ncl.ac.uk/Risks/`. 1985–2015.

[309] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[310] Paul Resnick and Richard Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebays reputation system. *The Economics of the Internet and E-commerce*, 11(2):23–25, 2002.

[311] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, pages 358–366, 1953.

[312] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[313] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[314] Jack Robertson and William Webb. *Cake-cutting algorithms: Be fair if you can*. AK Peters, 1998.

[315] David Robinson, Harlan Yu, and Aaron Rieke. Civil rights, big data, and our algorithmic future. Online. `https://bigdata.fairness.io/wp-content/uploads/2015/04/2015-04-20-Civil-Rights-Big-Data-and-Our-Algorithmic-Future-v1.2.pdf`, 2014.

[316] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99:1, pages 229–238, 1999.

[317] Franziska Roesner, Tadayoshi Kohno, and David Molnar. Security and privacy for augmented reality systems. *Communications of the ACM*, 57(4):88–96, 2014.

[318] Joan R Rosenblatt and James J Filliben. Randomization and the draft lottery. *Science*, 171(3968):306–308, 1971.

[319] Clinton Rossiter, John Jay, James Madison, and Alexander Hamilton. *The Federalist Papers: Alexander Hamilton, James Madison, John Jay; with an Introduction, Table of Contents, and Index of Ideas by Clinton Rossiter*. New American Library, 1788 (1961).

[320] Jean-Jacques Rousseau. *The Social Contract: & Discourses*. JM Dent & Sons, Limited, 1762 (1920).

[321] Cynthia Rudin. Algorithms for interpretable machine learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1519–1519. ACM, 2014.

[322] Mark D Ryan. Enhanced certificate transparency and end-to-end encrypted mail. *Proceedings of NDSS. The Internet Society*, 2014.

[323] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473. Springer, 2005.

[324] Ryuichi Sakai and Masao Kasahara. Id based cryptosystems with pairing on elliptic curve. *IACR Cryptology ePrint Archive*, 2003:54, 2003.

[325] Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[326] Christian Sandvig. Seeing the sort: The aesthetic and industrial defense of the algorithm. *Journal of the New Media Caucus*, 2015. ISSN: 1942-017X, [Online] `http://median.newmediacaucus.org/art-infrastructures-information/seeing-the-sort-the-aesthetic-and-industrial-defense-of-the-algorithm/`.

[327] Christian Sandvig, Kevin Hamilton, Karrie Karahalios, and Cedric Langbort. Auditing algorithms: Research methods for detecting discrimination on internet platforms. *Data and Discrimination: Converting Critical Concerns into Productive Inquiry*, 2014.

[328] Pierre Schlag. Rules and standards. *UCLA L. Rev.*, 33:379, 1985.

[329] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[330] Aaron Segal, Bryan Ford, and Joan Feigenbaum. Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2014.

[331] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[332] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – CRYPTO '84*, pages 47–53, 1984.

[333] Adi Shamir, Ronald L. Rivest, and Len Adleman. Mental poker. Technical Report LCS/TR-125, Massachussets Institute of Technology, Laboratory of Computer Science, 1979.

[334] shelat, a. and Shen, C. Two-output secure computation with malicious adversaries. *EUROCRYPT*, 2011.

[335] Victor Shoup. Practical threshold signatures. In *Advances in CryptologyEUROCRYPT 2000*, pages 207–220. Springer, 2000.

[336] Victor Shoup. Encryption algorithms – part 2: Asymmetric ciphers. ISO Standard 18033-2, May 2006. `www.shoup.net/iso/`.

[337] Madhavapeddi Shreedhar and George Varghese. Efficient fair queuing using deficit round-robin. *Networking, IEEE/ACM Transactions on*, 4(3):375–385, 1996.

[338] Daniel J Solove. A taxonomy of privacy. *University of Pennsylvania law review*, pages 477–564, 2006.

[339] Edith Spaan, Leen Torenvliet, and Peter van Emde Boas. Nondeterminism, fairness and a fundamental analogy. *Bulletin of the EATCS*, 37:186–193, 1989.

[340] Norton Starr. Nonrandom risk: The 1970 draft lottery. *Journal of Statistics Education*, 5(2), 1997.

[341] Jennifer G Steiner, B Clifford Neuman, and Jeffrey I Schiller. Kerberos: An authentication service for open network systems. In *USENIX Winter*, pages 191–202, 1988.

[342] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. Aegis: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 160–171. ACM, 2003.

[343] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, and Paul Pearce. Ad injection at scale: Assessing deceptive advertisement modifications. *Security and Privacy. IEEE*, 2015.

[344] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings Network and Distributed System Symposium*, 2010.

[345] Patrick P Tsang, Man Ho Au, Apu Kapadia, and Sean W Smith. Perea: Towards practical ttp-free revocation in anonymous authentication. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 333–344. ACM, 2008.

[346] Petar Tsankov, Srdjan Marinovic, Mohammad Torabi Dashti, and David Basin. Failsecure access control. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1157–1168. ACM, 2014.

[347] Michael Carl Tschantz, Amit Datta, Anupam Datta, and Jeannette M Wing. A methodology for information flow experiments. *Proceedings of the Computer Security Foundations Symposium (CSF)*, 2015.

[348] Michael Carl Tschantz, Amitava Datta, and Jeannette M Wing. Formalizing and enforcing purpose restrictions in privacy policies. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 176–190. IEEE, 2012.

[349] Michael Carl Tschantz, Anupam Datta, and Jeannette M Wing. Purpose restrictions on information use. In *Computer Security–ESORICS 2013*, pages 610–627. Springer, 2013.

[350] Michael Carl Tschantz, Dilsun Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems. *Electronic Notes in Theoretical Computer Science*, 276:61–79, 2011.

[351] Michael Carl Tschantz and Jeannette M. Wing. Formal methods for privacy. In *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, pages 1–15, 2009.

[352] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[353] Bureau of Consular Affairs U.S. State Department. Electronic diversity visa lottery. `https://www.dvlottery.state.gov/`.

[354] Jelle van den Hooff, M Frans Kaashoek, and Nickolai Zeldovich. Versum: Verifiable computations over large public logs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1304–1316. ACM, 2014.

[355] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[356] Vladimir Vapnik and Alexey Chervonenkis. A note on one class of perceptrons. *Automation and remote control*, 25(1), 1964.

[357] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *Proc. IEEE S & P*, 2013.

[358] Marc Waldman, Aviel D Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident censorship-resistant web publishing system. In *9th USENIX Security Symposium*, pages 59–72, 2000.

[359] Xueqiang Wang, Kun Sun, Yuewu Wang, and Jiwu Jing. Deepdroid: Dynamically enforcing enterprise policy on android devices. In *Proc. 22nd Annual Network and Distributed System Security Symposium (NDSS15). The Internet Society*, 2015.

[360] Samuel D Warren and Louis D Brandeis. The right to privacy. *Harvard law review*, pages 193–220, 1890.

[361] Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology–EUROCRYPT 2005*, pages 114–127. Springer, 2005.

[362] Brent R Waters, Dirk Balfanz, Glenn Durfee, and Diana K Smetters. Building an encrypted and searchable audit log. In *NDSS*, volume 4, pages 5–6, 2004.

[363] Robert NM Watson, Jonathan Woodruff, Peter G Neumann, Simon W Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, et al. Cheri: A hybrid capability-system architecture for scalable software compartmentalization. In *IEEE Symposium on Security and Privacy*, 2015.

[364] Barry R Weingast and Mark J Moran. Bureaucratic discretion or congressional control? regulatory policymaking by the federal trade commission. *The Journal of Political Economy*, pages 765–800, 1983.

[365] Daniel J Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Communications of the ACM*, 51(6):82–87, 2008.

[366] Xinyu Xing, Wei Meng, Dan Doozan, Nick Feamster, Wenke Lee, and Alex C Snoeren. Exposing inconsistent web search results with bobble. In *Passive and Active Measurement*, pages 131–140. Springer, 2014.

[367] Xinyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. Take this personally: Pollution attacks on personalized services. In *USENIX Security*, pages 671–686, 2013.

[368] Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. Unraveling the relationship between ad-injecting browser extensions and malvertising. In *Proceedings of The 24th International World Wide Web Conference (WWW)*, 2015.

[369] Andrew C Yao. Protocols for secure computations. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.

[370] Aydan R Yumerefendi and Jeffrey S Chase. Trust but verify: accountability for network services. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 37. ACM, 2004.

[371] Aydan R Yumerefendi and Jeffrey S Chase. The role of accountability in dependable distributed systems. In *Proceedings of HotDep*, volume 5, pages 3–3. Citeseer, 2005.

[372] Aydan R Yumerefendi and Jeffrey S Chase. Strong accountability for network storage. *ACM Transactions on Storage (TOS)*, 3(3):11, 2007.

[373] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness constraints: A mechanism for fair classification. *Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2015. `http://www.fatml.org/papers/Zafar_Valera_Gomez-Rodriguez_Gummadi.pdf`.

[374] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 325–333, 2013.

[375] Sheng Zhong, Jiang Chen, and Yang Richard Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1987–1997. IEEE, 2003.

[376] Yajin Zhou, Xiaoguang Wang, Yue Chen, and Zhi Wang. Armlock: Hardware-based fault isolation for arm. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 558–569. ACM, 2014.

[377] Phil Zimmerman. PGP users guide. *PGP Version*, 2(2), 1994.