MIT-CSAIL-TR-2007-040 July 27, 2007

Perfect Implementation of Normal-Form Mechanisms

Sergei Izmalkov, Matt Lepinski, and Silvio Micali

Perfect Implementation of Normal-Form Mechanisms

By Sergei Izmalkov, Matt Lepinski, and Silvio Micali[*]

July 25, 2007

Privacy and trust affect our strategic thinking, yet they have not been precisely modeled in mechanism design. In settings of incomplete information, traditional implementations of a normal-form mechanism —by disregarding the players' privacy, or assuming trust in a mediator— may not be realistic and fail to reach the mechanism's objectives. We thus investigate implementations of a new type.

We put forward the notion of a perfect implementation of a normal-form mechanism $\mathcal{M}$: in essence, an extensive-form mechanism exactly preserving all strategic properties of $\mathcal{M}$, *without relying on a trusted party or violating the privacy of the players.*

We prove that *any* normal-form mechanism can be perfectly implemented via envelopes and an envelope-randomizing device (i.e., the same tools used for running fair lotteries or tallying secret votes).

1. introduction

1.1. *The Privacy Problem of Implementing Normal-Form Mechanisms*

The game theoretic notion of a normal-form mechanism consists of a set of *reports* (or messages) for each player, and an *outcome function* mapping these reports to outcomes. For example, in the famous second-price auction mechanism, the reports consist of numerical bids —one for each player— and the outcome function returns the player with the highest bid as the *winner,* and the value of the second-highest bid as the *price.* Normal-form mechanisms can be designed so as to enjoy valuable theoretical properties. The characteristic property of the second-price mechanism is *efficiency:* because it is a dominant strategy for a player to bid his true valuation for the item for sale, at equilibrium the winner is the player with the highest true valuation.

Of course, however, normal-form mechanisms are *abstractions.* Outcome functions do not spontaneously evaluate themselves on players' reports. To be of use in concrete strategic settings, a normal-form mechanism must be *implemented*, but then its theoretical properties may suffer.

privacy vs. trust. An implementation of a normal-form mechanism is called *mediated* if its players rely on an external trusted party (the *mediator*), and *unmediated* otherwise. Every normal-form mechanism has implementations of both types; in particular, the following ones:

M_1 : (Mediated Mechanism) The players confide their reports to the mediator, who then secretly evaluates the outcome function and publicly announces the result.

[*]We would like to thank participants to numerous seminars, and especially Ran Canetti, Dino Gerardi, Sergiu Hart, Bart Lipman, and Dave Parks, for their comments. We are grateful to the NSF grant SES-0551244 for financial support.

M_2 : (Unmediated Mechanism) The players seal their reports into envelopes, which are then publicly opened so as to enable everyone to compute the outcome.[1]

These two general and concrete mechanisms are at opposite ends with respect to privacy and trust. Consider using them to implement a second-price auction in a *private-value* setting. On one hand, as long as the mediator is honest, M_1 reveals nothing more than the correct outcome. (However, the players cannot verify the mediator's honesty: nothing guarantees that he announces the correct outcome, or that he will keep their bids secret after the auction is over.[2]) On the other hand, M_2 guarantees the correctness of the outcome, but reveals much more: it makes the players' reports public knowledge in their entirety. In sum, M_1 requires total trust and offers total privacy, while M_2 requires no trust and offers no privacy.

PRIVACY AND TRUST AS A STRATEGIC PROBLEM. Other implementations may fall in between the above two extremes, but some loss of privacy appears to be unavoidable in unmediated mechanisms, and some reliance on trust unavoidable in mediated ones. Accordingly, privacy-valuing players may avoid participating in unmediated mechanisms, and distrustful players in mediated ones. In an auction, such reduced participation not only causes the seller to fetch lower prices, but negatively affects efficiency. Indeed, the requirement that "the item be won by the player who values it the most" applies to all potential bidders, not just the ones trusting a given mediator. Thus, no implementation that deters some players from bidding can be efficient in a general sense. In addition, whenever (for lack of other ways of getting a desired item) distrustful players participate in M_1 or privacy-valuing players participate in M_2, *neither mechanism can be efficient, even with respect to just the participating players.*

In M_1, a player who *truly distrusts* the mediator may fear that the price will always be artificially close to the winner's bid.[3] Such a player will not find it optimal to report his true valuation to the mediator; instead, he will have a strong incentive to "underbid." Thus, the item for sale might very well go to another player, who values it less but trusts the mediator more. Therefore, M_1 is not efficient.

In M_2, a player *truly valuing* the privacy of his valuation receives, *by definition,* some negative utility from its publicity. But then the only way for him to prevent his true valuation from becoming public is to bid a different value, perhaps in a randomized fashion. This distortion may thus make M_2 inefficient as well.

More generally, privacy and trust affect the players' strategic thinking, making it very hard for *concrete* mechanisms —whether mediated or unmediated— to satisfy the strategic properties of the *abstract* mechanisms they purportedly implement.

NOVELTY AND RELATIVE NATURE OF THE PROBLEM. In the second-price auction context, to preserve M_2's efficiency, one may be tempted to handle privacy-valuing players by (a) monetizing —somehow— privacy loss; (b) revising the players' valuations accordingly; and then (c) relying on the traditional Vickrey machinery. In so doing, however, the resulting auction would at best be

[1]For convenience, the players could employ an external party, who first collects their envelopes and then publicly opens them, without making the mechanism a mediated one. In fact, while a mediator is blindly trusted, such an external party only performs public actions, and thus all players can verify that he performs them correctly.

[2]According to Rothkopf, Teisberg, and Kahn (1990), such concerns explain why second-price auctions are rarely used in practice.

[3]If player i bids \$1000 dollars and the mediator (auctioneer) announces that i is the winner and must pay \$999 dollars, it is impossible for i to know if there was really a bid of \$999. Certainly, an auctioneer has incentives to manipulate the outcome (e.g., if he is paid a percentage of the generated revenue for his services) and to betray a player's secret (e.g., if he is offered a bribe for his disclosure).

efficient in the "revised" valuations, while Society's interest is that it be efficient in the "original" valuations. Integrating privacy and strategic concerns in the design of concrete implementations of normal-form mechanisms is a *new* problem, requiring new techniques and conceptual frameworks.

At a closer look, however, not even the *abstract* second-price auction itself may be perfectly efficient in a private-value setting. For instance, even in a magic world where the outcome function evaluates itself on players' reports, everyone is bound to learn that the winner's valuation is higher than the price; and this small loss of the winner's privacy may suffice to miss the target of efficiency in its purest form. But if the abstract second-price auction itself does not achieve perfect efficiency, what goal can we set for its concrete implementations? We answer this question with the strongest possible relativism: *they should always enjoy efficiency (or any other desired property) to exactly the same degree (whatever it may be) as the abstract second-price auction.*

More generally, any normal-form mechanism implies some loss of privacy. This is the case because the outcome itself (being a function of the players' reports) contains information about these reports. We regard this loss of privacy (which potentially affects all kinds of strategic concerns) as *inherent*, and do not wish to rectify or modify this —or any other— property of a normal-form mechanism.[4] That is: *in implementing a normal-form mechanism we aim at preserving exactly all its privacy properties.*

1.2. The Strategic Problem of Implementing Normal-Form Mechanisms

Together with a *context* $\mathcal{C}$ (describing, in particular, the players' preferences and beliefs), a normal-form mechanism $\mathcal{M}$ forms a *game* $\mathcal{G} = (\mathcal{M}, \mathcal{C})$, and $\mathcal{G}$'s *equilibria* are ultimately the object of interest. Therefore, equilibria constitute the fundamental measure of the quality of an implementation, $\mathcal{M}'$, of a normal-form mechanism $\mathcal{M}$.

At the most basic level, one may focus on a single equilibrium e of $\mathcal{G}$ and demand that $\mathcal{M}'$ be such that the game $\mathcal{G}' = (\mathcal{M}', \mathcal{C})$ has a payoff-equivalent equilibrium e'. More strongly, one may require that, for all equilibria of $\mathcal{G}$ of a specific type (e.g., dominant-strategy, ex post, Bayesian Nash, etc.), there are payoff-equivalent equilibria of $\mathcal{G}'$ of the same type. Stronger yet, $\mathcal{M}'$ should guarantee that $\mathcal{G}'$ *exactly* preserves *all* equilibria of $\mathcal{G}$ of *all* types; that is, $\mathcal{G}'$ should not loose nor modify any original equilibria, nor introduce any additional ones.[5]

In practice, the most transparent way in which an implementation of a normal-form mechanism $\mathcal{M}$ fails to preserve all equilibria of $\mathcal{M}$ is by providing extra strategic options to two or more players.[6] Consider the following "extreme" implementation of the second-price auction among 10 players.

Implementation $\widehat{\mathcal{M}}$*:* In a first step, all players simultaneously and privately submit their bids to players 9 and 10. In a second and last step, players 9 and 10 simultaneously announce the winner w and the price p. If the announced outcome is the same, then the good is sold as announced. Else, the good is not sold and all players are fined.

[4]When dealing with a sequence of normal-form mechanisms, however, a carefully designed loss of privacy may actually help to satisfy the strategic goals of the mechanisms yet to be played.

[5] In fact, Saijo, Cason, and Sjöström (2003) provide experimental evidence showing that even two games possessing the same set of dominant-strategy equilibria but a different set of Nash equilibria are actually played quite differently.

[6]Famous examples of auctions that resulted in outcomes that were not expected include FCC spectrum auctions, in which bidders used last few digits of their bids to communicate their preferences and to collude on the outcome (see Cramton and Schwartz (2002)) and the German GSM spectrum auction of 1999, in which two main participants, Mannesmann and T-Mobil, split the available 10 licenses at a very low price (see Grimm, Riedel, and Wolfstetter (2003)).

Clearly, $\widehat{\mathcal{M}}$ maintains the efficient equilibrium of the second-price mechanism —although, not in dominant strategies. However, unlike in the original normal-form mechanism, in implementation $\widehat{\mathcal{M}}$ players 9 and 10 *fully control the outcome:* indeed, they could choose, without recourse, any winner and price they want. In particular, based on the bids received in step 1, they can implement the second-price auctions only among themselves, that is, they could announce the winner to be the one between them with the highest bid, and the bid of the other as the price —which is individually incentive compatible. That is, implementation $\widehat{\mathcal{M}}$ brings into being a *new* equilibrium $\hat{e}$, that has no counterpart in the second-price, normal-form mechanism. This example illustrates that no concrete implementation $\mathcal{M}'$ of a normal-form mechanism $\mathcal{M}$ could be considered "perfect" without guaranteeing that $\mathcal{G}'$ and $\mathcal{G}$ enjoy equivalent sets of equilibria.

Another example suggests that we may want an even stronger property. Consider the following two-player normal-form games $\mathcal{G}_1$ and $\mathcal{G}_2$:

$\mathcal{G}_1$:

	A
A	$1,1$

$\mathcal{G}_2$:

	A	B
A	$1,1$	$101,0$
B	$0,101$	$100,100$

Note that $\mathcal{G}_2$ is a version of the Prisoner's Dilemma. Because (A,A) is its sole equilibrium, $\mathcal{G}_2$ trivially preserves all equilibria of $\mathcal{G}_1$. Intuitively, however, $\mathcal{G}_2$ is far from being perfectly "strategically equivalent" to $\mathcal{G}_1$. This is so because there is an outcome, namely (B,B), giving both players high payoffs but having no counterpart in $\mathcal{G}_1$. In some sense, the presence of such an outcome undermines the strength of equilibrium (A,A). For instance, although in this paper we do not concern ourselves with multiple interactions, if $\mathcal{G}_1$ and $\mathcal{G}_2$ were played repeatedly, we might expect outcome (B,B) to emerge when playing $\mathcal{G}_2$. Thus, a sequence of plays of $\mathcal{G}_1$ could yield vastly different outcomes than a sequence of plays of $\mathcal{G}_2$.

The above two examples indicate that, to ensure that playing a normal-form mechanism $\mathcal{M}$ is fully equivalent to playing a concrete implementation $\mathcal{M}'$ of $\mathcal{M}$, it is crucial that $\mathcal{M}$ *and* $\mathcal{M}'$ *enjoy the same strategic opportunities, not only for any single player but for any group of players as well.* Combining this strategic desideratum with the already discussed privacy one,

> *We define the problem of perfectly implementing a normal-form mechanism as that of perfectly matching all of its privacy and strategic properties.*

1.3. Our Contributions

Our contribution to a rigorous treatment of privacy and trust in mechanism design is as follows:

1. We introduce *ballot-box mechanisms,* a new class of unmediated, extensive-form mechanisms concretely playable via ballots and a ballot-box.

   Ballots and ballot-boxes are the same devices used, from time immemorial, for running fair lotteries and tallying secret votes. Such venerable devices have previously been part of many specific mechanisms, but today we demonstrate their power by showing that they can be used as universal implementation devices for normal-form mechanisms.

2. We put forward a rigorous definition of what it means for a ballot-box mechanism $\mathcal{B}$ to *implement perfectly* a mediated normal-form mechanism $\mathcal{M}$ in the classical setting.[7]

[7]By "classical setting" we mean a study of a single mechanism in isolation from any other interaction —concurrently

As we shall see, perfect implementation is a strong but technical notion that captures many a desideratum. In particular, whenever $\mathcal{B}$ perfectly implements $\mathcal{M}$, the following two main properties are guaranteed:

> *Strategic Equivalence.* For each player i, there is a one-to-one correspondence between his strategies in $\mathcal{B}$ and his strategies in $\mathcal{M}$, and the outcome of any profile of strategies in $\mathcal{B}$ is identical to the outcome of the profile of the corresponding strategies in $\mathcal{M}$.
>
> *Privacy Equivalence.* No set of players can gain more information (about the other players' strategies) in $\mathcal{B}$ than they can in $\mathcal{M}$, where the only information available to them coincides with what is deducible from the mechanism outcome.

3. We prove that *every* mediated normal-form mechanism is perfectly implementable by a ballot-box mechanism.

   In general, proving the existence of certain objects may not be helpful in finding them.[8] Our proof, instead, is constructive in a very strong sense. Indeed, we exhibit a *universal mechanism translator:* that is, a *single* algorithm that, given the description of *any* normal-form mechanism $\mathcal{M}$, *efficiently* outputs the description of a ballot-box mechanism perfectly implementing $\mathcal{M}$.

In particular, perfect implementation implies the exact preservation of all equilibria. Indeed, Strategic Equivalence states that a mediated mechanism $\mathcal{M}$ and its perfect implementation $\mathcal{M}'$ have identical (up to renaming and reordering of strategies) normal-form representations. And traditional equilibrium concepts (such as Nash, Bayesian, dominant strategy, ex post, undominated Nash, and trembling hand Nash equilibria) and set-valued solution concepts (such as rationalizability and iterated elimination of dominated strategies) are invariant to isomorphic transformations of normal-forms. Therefore, if a concrete mechanism $\mathcal{M}'$ perfectly implements a normal-form one $\mathcal{M}$, then, for all contexts $\mathcal{C}$ and all traditional solution concepts $\mathcal{E}$, the games $\mathcal{G} = (\mathcal{M}, \mathcal{C})$ and $\mathcal{G}' = (\mathcal{M}', \mathcal{C})$ are guaranteed to have the same set of $\mathcal{E}$-solutions.[9]

More generally, strategic equivalence and privacy equivalence guarantee that all subsets of players (including the set of all players) have the same strategic opportunities and have the same information in a play of a normal-form mechanism and in a play of its perfect implementation. Therefore, perfect implementations provide no additional incentives for any coalition of players —of any size— to form. Such bounding of coalitional power was first studied, in a slightly weaker sense, by Lepinski, Micali, Peikert, and Shelat (2004).

1.4. The General Nature of Our Contributions

A GENERAL NOTION. Although presented via ballot-box mechanisms, the notion of a perfect implementation is essentially independent of them. Ballot-box mechanisms are simply attractive because they work in a most intuitive manner and enable us to measure precisely the flow of

or in the future. We stress, however, that no restrictions are placed on the players' preferences. The players' beliefs (of any order) about anything relevant for the present as well as future payoffs are not limited in any way.

[8]For instance, Nash equilibria provably exist for any finite normal-form game, but how to find them efficiently remains an open problem.

[9]An example of a solution that is not normal-form invariant is a specific variation of a trembling hand equilibrium notion where the trembles (mistakes) are specified for *actions* (and not strategies) and are required to be equally likely for all available actions. This will lead to different distributions of trembles on normal-form strategies.

information during play (which is necessary for the rigorous treatment of privacy). What makes them unique, right now, is that they are universal: namely, a perfect implementation of *any* normal-form mechanism can be found within their class. We do conjecture, however, that other classes of concrete mechanisms capable of universal perfect implementation will be discovered.

A CONTEXT-FREE RESULT. A mechanism, considered in different contexts, yields games that may enjoy different properties. (For example, the unique Bayesian-Nash equilibrium of the first-price sealed-bid auction is efficient in the symmetric independent private-values setting, and not efficient in settings with any degree of asymmetry.) In principle, therefore, an automatic and universal procedure for perfect implementation might very well consist of an algorithm that, on input an arbitrary mechanism-context pair $(\mathcal{M}, \mathcal{C})$, outputs a concrete mechanism $\mathcal{M}'_{\mathcal{C}}$ "equivalent" to $\mathcal{M}$ in context $\mathcal{C}$. The applicability of such a procedure, however, would be quite limited: while $\mathcal{M}$ is a simple object (a set of messages and an outcome function), $\mathcal{C}$ may not be. For instance, as part of $\mathcal{C}$, the distribution on the players' types may be prohibitively complex to specify.

Our translator is instead *context-free.* Namely, on input an arbitrary normal-form mechanism $\mathcal{M}$, it generates a concrete mechanism $\mathcal{M}'$ such that, for all possible contexts $\mathcal{C}$, the players are indifferent between playing $\mathcal{M}$ and playing $\mathcal{M}'$ in context $\mathcal{C}$.

In essence, therefore, our translator achieves for mechanism implementation what Wilson (1987) advocates for mechanism design: namely, that *one should strive to find solutions minimizing dependence on the details of the problems at hand* (the "Wilson's Doctrine").

A SIGNAL-FREE SOLUTION. In a normal-form mechanism, players cannot signal information to one another. By contrast, signalling seems unavoidable in an extensive-form mechanism. Anytime a player A can choose between two or more actions that cause another player B to observe different consequences before having a choice of actions himself, there is an opportunity for A to communicate to B so as to affect the play and usher in new strategic opportunities. To guarantee strategic equivalence, we thus construct our perfect implementations to be signal-free. Technically, we ensure that the action set available to any player at any point of our ballot-box mechanisms consist of either (1) a single action, or (2) a pair of actions a_1 and a_2 generating identical observables for all other players. (In the second case, the player must make a choice, but this choice will only affect the final outcome, at which point no choice of actions is available to any player.)

COMPLEMENTING MECHANISM DESIGN. The practical meaningfulness of any abstraction depends on whether concrete implementations that adequately approximate it exist. In a sense, therefore, our contributions enhance the practical meaningfulness of the very notion of a normal-form mechanism: no matter how "delicate," its theoretical properties will continue to hold intact for at least one concrete implementation (i.e., its ballot-box implementation).

We view our results as *complementary* to mechanism design. As remarkable as they may be, the solutions offered by mechanism design are, most of the time, abstract normal-form mechanisms, which may not retain their properties when straightforwardly played by players who value privacy or do not trust anyone.[10] Thus, while we do not help a designer in engineering new mechanisms, by

[10]Sjöström and Maskin (2002) provide a comprehensive survey of mechanism design and implementation theory literature. Normal-form mechanisms are also extensively used in more applied fields, such as auction theory and contract theory, see Krishna (2002), Bolton and Dewatripont (2005). Often the problems of privacy and trust are by-passed by explicit additional assumptions. For instance, it is typically assumed that the seller (and similarly the principal) can fully commit to the mechanism she offers to the buyers (and similarly to the agents)—and, since buyers know that, their rationality dictates they must trust the seller.

perfectly implementing whatever abstract mechanisms he finds, we do enable him to ignore issues of privacy and trust in his work.

PERFECT IMPLEMENTATION OF PUBLIC- AND PRIVATE-OUTCOME MECHANISMS. In our discussion so far, we have emphasized normal-form mechanisms with *public outcome functions*, that is, mechanisms whose outcomes are totally and publicly revealed. More generally, however, mechanisms may also have *private outcome functions.* At the end on an execution of such a more general mechanism, not only a *public outcome* may be revealed to everyone, but each player may also privately learn a personal "piece of information", his *private outcome.*

We shall define and construct perfect implementations for these mechanisms as well. In particular, therefore, we perfectly implement second-price auctions in which only the winner and the seller learn the identity of the winner and the price, while all other players only learn that they did not win.

1.5. Prior Work on Privacy and Trust

ZERO KNOWLEDGE AND SIMULATORS. The study of privacy without trust started two decades ago in theoretical computer science; specifically, with the zero-knowledge proofs of Goldwasser, Micali, and Rackoff (1985). Assume that a prover P wishes (1) to prove a mathematical statement S to a verifier V, but also (2) to keep private any detail of the proof. Saying "S is true" is not convincing, and a classical proof of S, though convincing, would reveal much more than just "S is true".[11] In contrast, a zero-knowledge proof is an interactive process that enables P to convince any distrusting V that S is indeed true but *does not reveal any additional piece of knowledge.* Zero-knowledge proofs are formalized via the notion of a *simulator*, a conceptual tool that is also used in our paper.

SECURE COMPUTATION. A direct predecessor of our notion of perfect implementation is *secure computation,* as defined (and computationally achieved) by Goldreich, Micali, and Wigderson (1987), improving on a two-party result of Yao (1986). They showed that n parties, each possessing a secret input, can *securely evaluate any function* f on their inputs. That is, the n parties can talk back and forth so as to compute the output of f on their secret inputs with essentially the same correctness and privacy as if they privately handed their inputs to a trusted party, who would then secretly evaluate f and announce the result. Such privacy and correctness should hold even if some players deviate from their prescribed communication instructions. Specifically, secure computation protects against malicious *monolithic coalitions* — i.e., groups of perfectly coordinated players. To model the perfect coordination of a group of players C, secure computation envisages a single entity A, *the adversary,* that controls the members of C, before, during, and after the evaluation. (In particular, each message received by a member of the coalition C, during the communication protocol, is actually received by A, and any message sent by a member of C is actually chosen by A.) The way secure computation bounds the power of a monolithic coalition C is by guaranteeing that —as long as the players *not in* C stick to their communication instructions— whatever C can achieve in a secure evaluation of f (i.e., whatever influence C can have on f's output, and whatever knowledge C can gather about the inputs of the other players) the same C can achieve in a mediated evaluation of f.

[11] For instance, providing two large primes p and q whose product equals n is a proof of the statement $S =$ "n is the product of two primes." But such proof appears to contain much more knowledge than the mere statement "n is the product of two primes" (whatever they may be)!

Secure computation has been achieved in two main models of communication. In the first one —indeed, the one originally put forward by Goldreich, Micali, and Wigderson (1987)— the players communicate by ordinary broadcast, but use encryption to guarantee privacy. (The security properties of this model thus hold only for computationally bounded adversaries, who cannot crack such encryptions.) In the second one —put forward by Ben-Or, Goldwasser, and Wigderson (1988) and Chaum, Crépeau, and Damgård (1988)— the players communicate via perfectly private channels. That is, it is envisaged that between each pair of players i and j there exists a dedicated "lead pipe" that enable i and j to exchange messages so that no one else can see, alter, or gain any information about what they say to each other, or whether they are communicating at all. (The privacy guaranteed in this model is information theoretic, rather than computational.)

These two models differ in the size of the coalition they can tolerate. In the broadcast model, security is guaranteed against coalitions of any size. In the private-channel model, security is guaranteed only against coalitions of a *minority of the players.*[12]

Our paper is actually the first one to provide secure computation (via a different communication channel: namely ballots and a ballot box) in the information theoretic model and against coalitions of arbitrary size.

SECURE COMPUTATION VS. PERFECT IMPLEMENTATION. A protocol for securely evaluating a function f essentially is a concrete, extensive-form mechanism. It is thus very natural to consider implementing a normal-form mechanism $\mathcal{M}$ by the following concrete mechanism $\mathcal{M}^{SC}$: the players first choose their reports, and then securely compute on them the outcome function of $\mathcal{M}$. Unfortunately, this appealingly simple approach fails for three main reasons, only the first of which is correctable.

Honesty. The very notion of secure computation is stated relative to the honesty of some players. A honest player is one always following his instructions, and secure computation bounds the power of a coalition C only if the other players are honest. Honesty is clearly at odds with rationality: a rational player will surely deviate from his prescribed instructions whenever it is advantageous for him. Nonetheless, though explicitly part of the definition of secure computation, honesty by itself is only superficially problematic to the above mechanism $\mathcal{M}^{SC}$. For instance, in the secure-computation protocol of Goldreich, Micali, and Wigderson (1987), each player is required, for any encrypted message he sends, to give a (zero-knowledge) proof that the underlying clear-text message is actually in compliance with his prescribed strategy. If a player fails to provide such a proof, then he is immediately identified. Accordingly, if $\mathcal{M}^{SC}$ is enriched so that a sufficiently high fine is imposed on such a player, it will be rational for him to comply with his prescribed instructions.

Signaling. It is well know that enabling the players of a normal-form mechanism to signal to one another introduces additional equilibria, and all known secure-computation protocols enable *uncontrolled and undetected signaling* among the players —thus making $\mathcal{M}$ and $\mathcal{M}^{SC}$ strategically non equivalent. The possibility of signaling is self-evident in all secure-computation protocols in the private-channel model: by assumption, such channels allow for free and undetectable communication among the players. Uncontrolled signalling occurs more subtly in all secure-computation protocols in the broadcast model. In essence, these protocols require the players to execute probabilistic strategies, and thus exchange messages with positive "entropy." Whenever this is the case,

[12] Rabin and Ben-Or (1989) show that security can be guaranteed even against coalitions comprising $\lceil n/2 - 1 \rceil$ of the players; but a result of Cleve (1986) implies that this bound on coalition size is actually the highest possible.

Hopper, Langford, and von Ahn (2002) prove that any pair of players can implement a covert channel of communication, so called *steganographic communication*. That is, while exchanging their prescribed messages, any two players may also exchange additional information without anyone else (even a computationally unbounded observer) noticing it. By means of these covert channels, the players can signal to each other, and thus gain additional strategic opportunities not present in $\mathcal{M}$. The very inability of detecting such signalling rules out any possibility of punishing signalling players (so as to make it irrational to engage in steganographic communication). The only way to control signaling in secure computation is provided by the work of Lepinski, Micali, and Shelat (2005). Roughly, while they cannot prevent signaling in the secure computation of a function, they can ensure that such signaling is "no more powerful than a pre-play cheap-talk conversation." In essence, they show how to simulate the following 4-stage mediated mechanisms: (1) the players freely engage in cheap talk (but cannot contract with each other); (2) each player, now locked in his own room, chooses his report and submits it to the mediator; (3) the mediator applies the outcome function to the submitted reports; and (4) the mediator broadcasts the public outcome, and privately delivers to each player his private outcome. Though this is a very powerful simulation, the initial cheap-talk stage significantly alters the strategic opportunities of a normal-form mechanism. In sum, no implementations of normal-form mechanisms based on known secure-computation protocols can satisfy our notion of strategic equivalence.

Coalitions. A separate source for the strategic non equivalence of $\mathcal{M}$ and $\mathcal{M}^{SC}$ is the "excessive" power enjoyed by some coalitions of players. Indeed, for all secure-computation protocols there exists at least one coalition C that not only can *force any outcome it wants* (as the players 9 and 10 can do for the implementation $\widehat{\mathcal{M}}$ of the second-price mechanism discussed in Section 1.2), but can do so *without being detected by anyone*.[13] In the protocol of Goldreich, Micali, and Wigderson (1987), only the coalition of all players has such power. In all secure-computation protocols that rely on private channels, C can be any coalition comprising a majority of the players. Such large coalitions may not be a concern if honesty is assumed to be the norm, but they are very much a concern when all players are assumed to be rational.

Our paper can interpreted as the first one to guarantee secure computation based solely on rationality of the players (no matter of the incentive structure and of size of a coalition).[14]

1.6. Other Related Work

Privacy of information was certainly recognized in earlier economics papers, but from a normative perspective, not one of mechanism design.[15] Closer to our work, trusted-party simulation has further been investigated in the following three directions.

[13]In all prior secure-computation protocols, such a coalition can force any outcome it wants with the guarantee that no player not in the coalition (let alone an external observer!) can notice any wrong doing. This is not the case if players 9 and 10 collude in $\widehat{\mathcal{M}}$. For instance, if player 1 reports a bid of 100, and players 9 and 10 announce that the winner is player 9 and the price is 5, then player 1 immediately realizes that 9 and 10 have cheated.

[14]Indeed, Izmalkov, Lepinski, and Micali (2005) cast an earlier version of our results in terms of secure computation.

[15]The special issue "The Law and Economics of Privacy" of The Journal of Legal Studies (1980, Vol. 9(4)) and Posner (1981) address the question of and under which circumstances should private information be collected and publicly released. Stigler (1980) provides several examples of economic phenomena that are linked to privacy concerns: reputation, blackmail, industrial secrecy and espionage.

SPECIAL CASES. All the literature on pre-play communication games achieving correlated equilibrium (or communication equilibrium, or Bayesian-Nash equilibrium) can be viewed as replacing the mediator in the evaluation of *a specific class of probabilistic functions* f. (See in particular, Bárány (1992), Forges (1990), Ben-Porath (1998), Aumann and Hart (2003), Urbano and Vila (2002), Ben-Porath (2003), Gerardi (2004), Dodis, Halevi, and Rabin (2000), Gerardi and Myerson (2005), Krishna (2006).) In addition, the emphasis of all these papers is on preserving specific equilibria of the original game, without concerns about other equilibria that may be generated in the process.

Halpern and Teague (2004) —see also Gordon and Katz (2006) and Lysyanskaya and Triandopoulos (2006)— use secure function evaluation to replace the mediator *for a specific class of incentives.* (In essence, they require that the preferences are such that the function to be evaluated is dominant-strategy incentive compatible.) Again, no consideration is given to the introduction of additional equilibria.

By contrast, we replace the mediator *for all functions* and *for all possible preferences* of the players, and *keep all equilibria intact.*

SPREADING TRUST. Some works, in particular that of Naor, Pinkas, and Sumner (1999), rather than putting trust on a single mediator, distribute it onto multiple mediators. (Thus, correctness and privacy hold only in so far these mediators do not collude with each other, nor signal information that they are not supposed to.)

By contrast, our emphasis is on *removing all trusted parties.*

IMPOSSIBILITY RESULTS. Whether or not a trusted mediator can be replaced by an unmediated interaction of the players alone crucially depends on the means of interaction available to the players. Because such a replacement is counter-intuitive, we informally expect it to be impossible in most interaction models. Indeed, this replacement has been *proved* impossible, in a formal sense, in many specific interaction models, *even for a restricted class of contexts and outcome functions.* Notably, Aumann and Hart (2003) prove that two players cannot reach any non-trivial *correlated equilibrium* via "cheap talk," so long as the players communicate essentially by broadcasting messages. Brandt and Sandholm (2004) argue the impossibility of *unconditionally privacy-preserving second-price auctions* in many interaction models.

By contrast, we prove that *there exists a reasonable model of interaction* (via ballots and a ballot box) in which replacing the mediator is possible *for all outcome functions and all contexts.*

1.7. Road Map

The rest of the main body of this paper intuitively describes our notions and results for normal-form mechanisms with public outcomes. Their formalizations, and our proofs, are presented in Sections A through F of our Appendix. Section G extends our work to normal-form mechanisms with both public and private outcomes. Finally, Section H discusses the issue of *abort*, that is, the players' ability (in certain settings) of taking no actions rather than those specified in their action sets.

2. THE INTUITIVE NOTION OF A BALLOT-BOX MECHANISM

Ballot-box mechanisms are extensive-form, imperfect-information mechanisms with Nature. Accordingly, to specify them we must specify who acts when, the actions and the information available to the players, when the play terminates, and how the outcome is determined upon termination.

A ballot-box mechanism ultimately is a mathematical abstraction, but possesses a quite natural physical interpretation. The physical setting is that of a group of players, seated around a table, acting on a set of *ballots* with the help of a randomizing device, the *ballot-box.* Within this physical setting, one has considerable latitude in choosing natural actions available to the players. In this paper, we make a specific choice, sufficient for our present goals. In future papers, one may make different choices, so as to achieve different goals, and still deal with a ballot-box mechanism.

BALLOTS. There are two kinds of ballots: *envelopes* and *super-envelopes.* Externally, all ballots of the same kind are identical, but super-envelopes are slightly larger than envelopes. An envelope may contain a symbol from a finite alphabet, and a super-envelope may contain a sequence of envelopes. (Our constructions actually needs only envelopes containing an integer between 1 and 5, and super-envelopes capable of containing at most 5 envelopes.) An envelope perfectly hides and guarantees the integrity of the symbol it contains until it is opened. A super-envelope tightly packs the envelopes it contains, and thus keeps them in the same order in which they were inserted. Initially, all ballots are empty and in sufficient supply.

BALLOT-BOX ACTIONS. There are 8 classes of actions in a ballot-box mechanism —those in the first 7 classes are taken by the players, and those in the eighth class are taken by Nature.

Each action in the first 6 classes is referred to as a *public action*, because a player performs it in plain view, so that all players know exactly which action has been performed. These classes are: (1) publicly writing a symbol on a piece of paper and sealing it into a new, empty envelope; (2) publicly opening an envelope to reveal its content to all players; (3) publicly sealing a sequence of envelopes into a new super-envelope; (4) publicly opening a super-envelope to expose its inner envelopes; (5) publicly reordering a sequence of envelopes; and (6) publicly destroying a ballot. The last two classes of public actions are not actually necessary, but considerably simplify the description of our construction. Note that each public action has the same effects, no matter which player performs it.

An action in the seventh class is referred to as a *private action,* because a player performs it outside of public view, so that the other players do not know which action has been performed. There are two possible private actions for each pair of ballots of the same kind (i.e., either two envelopes or two super-envelopes containing the same number of envelopes). To perform the 0-action, a player i picks up both ballots, hides them behind his back, and then returns them to the table in the same order. To perform the 1-action, i picks up both ballots, hides them behind his back, and then returns them in the opposite order. Because the other players cannot tell whether the order has been changed, i is effectively choosing a secret bit. We call two private actions *complementary* if they are the 0-action and 1-action for the same pair of ballots.

An action in the eighth class is referred to as an *action of Nature.* Such an action consists of "ballot boxing" a publicly chosen sequence of ballots. That is, it consists of reordering the chosen ballots according to a permutation randomly chosen by —and solely known to— Nature.

PUBLIC AND PRIVATE INFORMATION. The players can keep track of the ballots on the table. We formalize this ability by associating to each ballot a unique identifier: a positive integer that is common information to all players. These identifiers correspond to the order in which the ballots are placed on the table for the first time, or returned to the table after being picked up and permuted —by the ballot box, by a private action, or by a public reordering.

The public and private information generated by ballot-box actions are intuitively described as follows. Each action, when played, generates a publicly available string. In the case of a public action A on ballots j, k, l,..., this string consists of "A, j, k, l, ...". A private action additionally generates a secret bit available only to the acting player. If player i secretly re-orders ballots j and k, then the public string consists of "a re-ordering of ballots j and k has occurred," and i's secret bit corresponds to the specific re-ordering he chose. The *public record* is the concatenation of the public strings generated by all actions played thus far. Similarly, for each player i, *i's private record* is the concatenation of the bits corresponding to all private actions played by i.

MECHANISM PLAY. A ballot-box mechanism is played in a fixed number of rounds. Players and Nature act one at a time, in a pre-specified order. In each round, the information available to the acting player consists of the current public record and his current private record, and the actions available to him are described by a pre-specified function of the current public record.

The mechanism terminates when all players in the fixed sequence have acted, and the outcome is a pre-specified function of the final public record.

3. THE INTUITIVE NOTION OF A PERFECT IMPLEMENTATION

3.1. *Standard Normal-Form Mechanisms and Their Logical Structure*

All normal-form mechanisms considered in this paper are finite; that is, have finitely many possible messages and outcomes. For convenience, we actually focus on *standard normal-form mechanisms;* that is, normal-form mechanisms where all messages and outcomes have the same length. (This is without any loss of generality, since messages and outcomes could always be artificially padded so as to be equally long.)

We view such abstract mechanisms as been played in three conceptual stages: (1) an initial *commitment stage,* in which each player —independently of the others— selects his message string; (2) a magical *computation stage,* in which the outcome function spontaneously evaluates itself on the selected messages; and (3) a final *revelation stage,* in which the resulting outcome is announced.

3.2. *A Structural Definition of Perfect Implementation*

At a high level, a *perfect implementation* of a standard normal-form mechanism $\mathcal{M}$ is defined to be a ballot-box mechanism $\mathcal{B}$ that preserves the above logical structure of $\mathcal{M}$. (We shall then prove that such a structural definition guarantees that $\mathcal{B}$ is strategically and privacy equivalent to $\mathcal{M}$.) Let us now break down this high-level definition into its syntactic and semantic components.

Syntactically, $\mathcal{B}$ consists of 3 *ballot-box protocols,*[16]corresponding to the 3 conceptual stages of $\mathcal{M}$: a *ballot-box committer,* a *ballot-box computer,* and a *ballot-box revealer.* These protocols, letting $\mathcal{M}$'s messages be L-bit long and letting g be the outcome function, have the following functionalities.

1. The ballot-box committer is executed first and results in each player i having an input m_i. The players are free to choose whatever inputs they like, but are then "committed" to them: by the end of the stage, for each player i, there is a separate sequence of envelopes whose contents encode m_i.

2. The ballot-box computer is executed next and transforms the envelopes encoding $m_1, \ldots, m_n$ into envelopes encoding the outcome $g(m_1, \ldots, m_n)$.

[16]Ballot-box protocols essentially are ballot-box mechanisms which may start with a non-empty set of ballots, or produce no outcome.

3. The ballot-box revealer is executed last and solely consists of opening the envelopes whose contents encode the outcome: by decoding these now public contents, each player individually learns $g(m_1, \ldots, m_n)$.

Semantically, $\mathcal{B}$'s protocols satisfy the following properties.

Minimum Information. The first two protocols reveal no information about the inputs; and the third one reveals only the outcome. (This information requirement matches that of $\mathcal{M}$.)

Minimum Choice. In the first protocol of $\mathcal{B}$, for each player i, there are exactly L rounds (corresponding to the L bits of i's message) in which i is active and his action set consists of two complementary private actions. In all other rounds of $\mathcal{B}$, the action set consists of a *single* action: either a public one or an action of Nature. (This requirement matches the fact that in $\mathcal{M}$ each player i chooses his own L- bit message m_i and makes no other strategic choices.)

Same Bit-by-Bit Encoding. Of course, all three protocols must use the same encoding scheme, which encodes a binary string by concatenating the encodings of its individual bits. (This technical requirement matches the fact that the players and the mediator of $\mathcal{M}$ "talk in the same language", and in this language a message is the concatenation of its individual bits.)

3.3. Perfect Implementation Implies Strategic and Privacy Equivalence

Let us now briefly explain two crucial semantic implications of the above structural definition.

Informally, an extensive-form (and imperfect-information) mechanism $\mathcal{B}$ is *strategically equivalent* to a standard normal-form mechanism $\mathcal{M}$ if "there exist isomorphisms between the players' strategies in the two mechanisms that preserve the outcomes." That is, no matter which strategies the players may select in $\mathcal{B}$, they are guaranteed to have corresponding strategies in $\mathcal{M}$ that yield exactly the same outcomes. Viceversa, for each possible way to play $\mathcal{M}$, the players are guaranteed to have a corresponding way to play $\mathcal{B}$ which produces an identical outcome. Importantly, this correspondence is not between strategy profiles, but between the strategies of each *individual* player. That is, every player i can "translate" his own strategy from one mechanism to the other without any cooperation from the others. Thus, the players truly are strategically indifferent between playing $\mathcal{M}$ or $\mathcal{B}$.

Assuming that a ballot-box mechanism $\mathcal{B}$ is strategically equivalent to a standard normal-form mechanism $\mathcal{M}$, we further say that $\mathcal{B}$ is *privacy equivalent* to $\mathcal{M}$ if, for any subset P of the players and any strategy profile s, the sequences of decision nodes the players in P go through in an execution of $\mathcal{B}$ with profile s, can be generated from just the final outcome and s_P, the strategy sub-profile of P.[17] This technical property can be interpreted as follows. At the end on an execution of $\mathcal{M}$, the players of P collectively know: (1) the strategies they used (i.e., the messages they sent to the mediator) and (2) the final outcome. At the end of an execution of $\mathcal{B}$ the players of P collectively know: ($1'$) the strategies they used and ($2'$) the sequences of decision nodes they went through.[18] Because $\mathcal{B}$ is strategically equivalent to $\mathcal{M}$, ($1'$) and ($2'$) suffice to reconstruct

[17]Actually we prove a much stronger property when $\mathcal{B}$ is a perfect ballot-box implementation of $\mathcal{M}$: informally, that there exists a probabilistic algorithm that on just the outcome information —without any knowledge of the total profile of strategies s that generated it— reproduces a "fictitious" public record with the same probability distribution with which s would have generated it.

[18]Note that, due to the strategic equivalence of $\mathcal{M}$ and $\mathcal{B}$, (1) and ($1'$) are actually equivalent.

the outcome, but in principle could contain *additional* information. However, when $\mathcal{B}$ is privacy equivalent to $\mathcal{M}$ they cannot. Therefore, all players are indifferent between playing $\mathcal{M}$ and $\mathcal{B}$ also from a privacy perspective.

We formalize strategic and privacy equivalence in our appendix, and then prove the following.

THEOREM 1: *If a ballot-box mechanism $\mathcal{B}$ perfectly implements a standard normal-form mechanism $\mathcal{M}$, then $\mathcal{B}$ is strategically and privacy equivalent to $\mathcal{M}$.*

3.4. Existence of Perfect Implementations

In light of our sketched definition of a perfect implementation, to prove the existence of a perfect implementation for general standard normal-form mechanisms, we must prove the existence of general ballot-box committers, computers and revealers. Proving the existence of general ballot-box committers is not a problem: after finding a procedure to commit to a single bit, repeating it L times enables the players to commit to L bits. Ballot-box revealers present no problems either: a revealer simply consists of opening a fixed sequence of envelopes. Proving the existence of ballot-box computers, however, is a much bigger problem, and we can only solve it in a modular fashion.

Modularity enables one to understand and to design complex systems by breaking them into a multiplicity of simpler components, and is thus central to Science and Engineering.[19] In our case, it enables us to solve our problem as follows: first, we construct ballot-box computers for a few elementary functions, then we show that it is possible to "assemble" them so as to obtain a ballot-box computer for any desired finite function. The net result is summarized in the following result, proved in our appendix.

THEOREM 2: *For every standard normal-form mechanism $\mathcal{M}$, there exists a ballot-box mechanism $\mathcal{B}$ that perfectly implements $\mathcal{M}$.*

3.5. Efficient Construction of Perfect Implementations

In principle, the existence of a ballot-box mechanism perfectly implementing a mediated one may be highly non-constructive. In such a case, finding a perfect implementation would require a new ad hoc construction for each mediated mechanism of interest. We dispel this possibility by providing a *universal method* of finding perfect ballot-box implementations. That is, we exhibit a single "compiling" algorithm $\mathbb{C}$ converting (the description of) any standard normal-form mechanism into (the description of) a ballot-box mechanism perfectly implementing it.

Such a universal method, however, would be practically useless if it were computationally infeasible to carry out. Traditionally, players' resources (for reasoning and computing) are assumed to be unbounded in game theory, but bounded in cryptography. Our construction, however, achieves the best of both worlds. Namely, *perfect implementation applies to players with unlimited resources, but is achievable by players with limited ones.* For any standard normal-form mechanism $\mathcal{M}$, not only do ballot-box mechanisms perfectly implementing $\mathcal{M}$ exist, but ours are also *(1) very easy to find, and (2) very easy to play.* Both properties are formalized in our appendix, where we prove the following result.

[19]In Science, the usefulness of modularity is eloquently exemplified by the extensive use of lemmas in long proofs. As for Engineering, the very computer used in typesetting this paper has been manufactured by integrating together hundreds of individual components, often produced by separate firms.

Theorem 3: *There exists a linear-time algorithm $\mathbb{C}$ that, on input any standard normal-form mechanism $\mathcal{M}$, outputs a linear-time ballot-box mechanism $\mathcal{B}$ that perfectly implements $\mathcal{M}$.*

3.6. The Key to Our Universal Construction.

As for the earlier constructions of Gödel and Turing, in logic and computation respectively, the universality of ours stems from the ability of "equating subjects and objects." In the case of Gödel numberings, for example, the coherence of Peano's arithmetics is itself encoded as a sentence in such arithmetics. In our case, we represent data and operations as permutations of the integers 1 through 5 as follows: each permutation p is encoded into a sequence of 5 envelopes, so that envelope j contains $p(j)$. Such a sequence of envelopes is both a piece of (really physical!) *data,* as well as an *algorithm*, which via the ballot box is capable of operating on other data. For instance, one of the crucial subroutines of our construction is a procedure that, given two sequences of 5 envelopes —the first sequence encoding a permutation p and the second a permutation q— interprets the first sequence as a multiplication program and returns (without revealing any information about p or q) a sequence of 5 envelopes encoding the product permutation pq, that is, the permutation mapping each i between 1 and 5 to $p(q(i))$.

4. extensions and conclusions

In this paper, we have shown that any *single* normal-form mechanism can be implemented without altering its strategic and privacy properties. In a forthcoming paper, we generalize the present results so as to implement perfectly *multiple* mediated mechanisms. Such mechanisms may also be interdependent in an arbitrary way: that is, they can be executed simultaneously or sequentially, and their mediators may privately communicate with each other.[20] Achieving these results will require a more general treatment of modularity in mechanism design.

These results are just a start, and much more needs to be done. People care about privacy. And to make more accurate predictions and develop more meaningful models, privacy should intrinsically inform any general study of human interactions. In particular, we believe and hope that a rigorous and comprehensive treatment of privacy will become an an integral part of game theory.

Department of Economics, Massachusetts Institute of Technology, 50 Memorial Drive, E52-252C, Cambridge, MA 02142; izmalkov@mit.edu;

BBN Technologies, 10 Moulton Street, Cambridge, MA 02138; mlepinski@bbn.com;

and

Computer Science and Artificial Intelligence Laboratory, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 32 Vassar Street, Room 32-G644, Cambridge, MA 02139; silvio@csail.mit.edu.

APPENDIX

[20] For instance, in implementing two auctions one after the other, the first mediator may announce the winner and the second highest bid, but pass on to the second mediator some aggregate information of all bids —e.g., the average of all bids, which may be used to set the reserve price of the second auction.

A. Notation

Basics. We denote by $\mathbb{R}^+$ the set of non-negative reals; by Σ the alphabet consisting of English letters, arabic numerals, and punctuation marks; by Σ^* the set of all finite strings over Σ; by $\perp$ a symbol not in Σ; by $\mathbb{S}_k$ the group of permutations of k elements; by $x := y$ the operation that assigns value y to variable x; by $\emptyset$ the empty set, and by ϕ the empty string/sequence/vector.

If S is a set, by S^0 we denote the empty set, and by S^k the Cartesian product of a set S with itself k times.

If x is a sequence, by either x^i or x_i we denote x's ith element,[21] and by $\{x\}$ the set $\{z : x^i = z \text{ for some } i\}$. If x is a sequence of k elements and i and j are positive integers such that $i < j \leq k$, then by $x^{[i,j]}$ and $x^{(i,j]}$ we respectively denote the sequences $x^i, \ldots, x^j$ and $x^{i+1}, \ldots, x^j$. If x is a sequence of k integers, and m is an integer, by $x + m$ we denote the sequence $x^1 + m, \ldots, x^k + m$. If x and y are sequences, respectively of length j and k, by $x \circ y$ we denote their concatenation (i.e., the sequence of $j + k$ elements whose ith element is x^i if $i \leq j$, and y^{i-j} otherwise). If x and y are strings (i.e., sequences with elements in Σ), we denote their concatenation by either xy or $x|y$.

Players and profiles. We always denote by N the (finite) set of players, and by n its cardinality. If i is a player, $-i$ denotes the set of the other $n-1$ players, that is, $-i = N \setminus \{i\}$. Similarly, if $C \subset N$, then $-C$ denotes $N \setminus C$. A profile is a vector indexed by N. If x is a profile, then, for all $i \in N$ and $C \subset N$, x_i is i's component of x and x_C is the sub-profile of x indexed by C; thus: $x = (x_i, x_{-i}) = (x_C, x_{-C})$.

Probability distributions. All distributions considered in this paper are over finite sets. If $X : S \to \mathbb{R}^+$ is a distribution over a set S, we denote its support by $[X]$, that is, $[X] = \{s \in S : X(s) > 0\}$. We consider two distributions X and Y equal if $[X] = [Y]$ and $X(s) = Y(s)$ for all $s \in [X]$. We denote by $rand(S)$ the uniform distribution over S.

If X is a distribution, $x \leftarrow X$ denotes the operation that assigns to variable x an element of $[X]$ selected according to X. If $f, g, \ldots$ are functions and $X, Y, \ldots$ are distributions, by

$$\langle\, f(x, y, \ldots), g(x, y, \ldots), \ldots \,:\, x \leftarrow X; y \leftarrow Y \ldots \rangle$$

we denote the distribution generated by the following experiment: first select x according to X, second select y according to Y, and so on; then return the sequence $f(x, y, \ldots), g(x, y, \ldots), \ldots$.

If A is a probabilistic algorithm, the distribution over A's outputs on input x is denoted by $A(x)$. A probabilistic function $f : X \to Y$ is *finite* if X and Y are both finite sets and, for every $x \in X$ and $y \in Y$, the probability that $f(x) = y$ has a finite binary representation.

B. Normal-form mechanisms

Definition 1: A *normal-form mechanism* is a tuple $\mathcal{M} = (N, M, Y, g)$, where:

- N, the *set of players*, is a finite set;
- M, the *message space*, is the Cartesian product of n subsets of Σ^*, $M = M_1 \times \cdots \times M_n$;
- Y, the *outcome set*, is a subset of Σ^*; and
- g, the *outcome function*, is a probabilistic function, $g : M \to Y$.

[21] For any given sequence, we shall solely use superscripts, or solely subscripts, to denote all of its elements.

We refer to each M_i as the *message space* (or *strategy set*) of player i, and to each $m \in M_i$ as a *message* (or *strategy*) of i. $\mathcal{M}$ is *finite* if M and Y are finite sets and g is a finite function.

Definition 2: A normal-form mechanism $\mathcal{M} = (N, M, Y, g)$ is a *standard normal-form mechanism* if $M_1 = \cdots = M_n = Y = \{0,1\}^L$ for some integer L; that is, $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$.

remark. Without loss of generality, we focus solely on standard normal-form mechanisms.[22]

C. ballot-box mechanisms

Definition 3: An *envelope* is a triple $(j, c, 0)$, where j is a positive integer, and c a symbol of Σ. A *super-envelope* is a triple (j, c, L), where both j and L are positive integers, and $c \in \Sigma^L$. A *ballot* is either an envelope or a super-envelope. If (j, c, L) is a ballot, we refer to j as its *identifier*, to c as its *content*, and to L as its *level.* (As we shall see, L represents the number of inner envelopes contained in a ballot.)

Let B be a set of ballots and j an integer. Then, by $cont_B(j)$ we denote the symbol $c \in \Sigma$, if B has a single envelope with identifier j and content c, and the symbol $\perp$ otherwise. If j is a sequence of positive integers, $j = j^1, \ldots, j^m$, then by $cont_B(j)$ we denote the symbol $\perp$ if $cont_B(j^k) =\perp$ for some element j^k, and the string $cont_B(j^1)|\cdots|cont_B(j^m)$ otherwise.

A set of ballots B is *well-defined* if distinct ballots have distinct identifiers. If B is a well-defined set of ballots, then I_B denotes the set of identifiers of B's ballots. For $j \in I_B$, B_j (or the expression *ballot j*) denotes the unique ballot of B whose identifier is j. For $J \subset I_B$, B_J denotes the set of ballots of B whose identifiers belong to J. To emphasize that ballot j actually is an envelope (super-envelope) we may use the expression *envelope j* (*super-envelope j*).

Definition 4: A *global memory* for a set of players N consists of a triple (B, R, H), where

- B is a well defined set of ballots;
- R is a sequence of strings in Σ^*, $R = R^1, R^2, \ldots$; and
- H is a profile of sequences of strings in Σ^*, $H = H_1, \ldots, H_n$.

We refer to B as the *ballot set;* to R as the *public record;* to each element of R as a *record;* to H as the *private history* profile; and to each H_i as the *private history of player i*. The *empty global memory* is the global memory for which the ballot set, the public record, and the private history of every player are all empty. We denote the set of all possible global memories by GM.

Definition 5: Ballot-box actions are functions from GM to GM. The subset of ballot-box actions available at a given global memory gm is denoted by $\mathcal{A}_{gm}$. The actions in $\mathcal{A}_{gm}$ are described below, grouped in 8 classes. For each $a \in \mathcal{A}_{gm}$ we provide a formal identifier; an informal reference (to facilitate the high-level description of our constructions); and a functional specification. If $gm = (B, R, H)$, we actually specify $a(gm)$ as a program acting on variables B, R, and H. For convenience, we include in R the auxiliary variable `ub`, the *identifier upper-bound*: a value equal to 0 for an empty global memory, and always greater than or equal to any identifier in I_B.

[22] Indeed, any finite, mediated normal-form mechanism $\mathcal{M} = (N, M, Y, g)$ can be put into standard form as follows. Let z be the cardinality of the largest set among $M_1, \ldots, M_n$, and Y. Choose $L = \lceil \log(z) \rceil$. Letting c_i be the cardinality of message set M_i, encode the elements of M_i as the lexicographically first c_i strings in $\{0,1\}^L$ and consider any string $x \in \{0,1\}^L$ lexicographically greater than c_i as an alternative encoding of the first element of M_i. Analogously encode the outcome set Y as elements of $\{0,1\}^L$. Define now $g' : (\{0,1\}^L)^n \to \{0,1\}^L$ to be the following function: if x'_i is an encoding of $x_i \in M_i$ for all $i \in N$, and if y' is an encoding of $y \in Y$, then $g'(x'_1, \ldots, x'_n) = y'$ if and only if $g(x_1, \ldots, x_n) = y$. Then, $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g')$ is a standard normal-form mechanism equivalent to $\mathcal{M}$.

1. $(\textsc{NewEn}, c)$ —where $c \in \Sigma$.

   "Make a new envelope with public content c."

   $\mathtt{ub} := \mathtt{ub} + 1$; $B := B \cup \{(\mathtt{ub}, c, 0)\}$; and $R := R \circ (\textsc{NewEn}, c, \mathtt{ub})$.

2. $(\textsc{OpenEn}, j)$ —where j is an envelope identifier in I_B.

   "Publicly open envelope j to reveal content $cont_B(j)$."

   $B := B \setminus \{B_j\}$ and $R := R \circ (\textsc{OpenEn}, j, cont_B(j), \mathtt{ub})$.

3. $(\textsc{NewSup}, j_1, \ldots, j_L)$ —where $L \leq 5$, and $j_1, \ldots, j_L \in I_B$ are distinct envelope identifiers.

   "Make a new super-envelope containing the envelopes $j_1, \ldots, j_L$."

   $\mathtt{ub} := \mathtt{ub} + 1$; $B := B \cup \{(\mathtt{ub}, (cont_B(j_1), \ldots, (cont_B(j_L)), L)\}$;
   $B := B \setminus \{B_{j_1}, \ldots, B_{j_L}\}$; and $R := R \circ (\textsc{NewSup}, j_1, \ldots, j_L, \mathtt{ub})$.

4. $(\textsc{OpenSup}, j)$ —where $j \in I_B$ is the identifier of a super-envelope of level L.[23]

   "Open super-envelope j."

   letting $cont_B(j) = (c_1, \ldots, c_L)$, $B := B \cup \{(\mathtt{ub} + 1, c_1, 0), \ldots, (\mathtt{ub} + L, c_L, 0)\}$; $B := B \setminus \{B_j\}$;
   $\mathtt{ub} := \mathtt{ub} + L$; and $R := R \circ (\textsc{OpenSup}, j, \mathtt{ub})$.

5. $(\textsc{PublicPermute}, j_1, \ldots, j_k, p)$ —where $k \leq 5$, $p \in SYM_k$, and $j_1, \ldots j_k \in I_B$ are distinct identifiers of ballots with the same level L.

   "Publicly permute $j_1, \ldots, j_k$ according to p."

   $B := B \cup \{(\mathtt{ub} + 1, cont_B(j_{p(1)}), L), \ldots, (\mathtt{ub} + k, cont_B(j_{p(K)}), L)\}$; $B := B \setminus \{B_{j_1}, \ldots, B_{j_k}\}$;
   $\mathtt{ub} := \mathtt{ub} + k$; and $R := R \circ (\textsc{PublicPermute}, j_1, \ldots, j_k, p, \mathtt{ub})$.

6. $(\textsc{Destroy}, j)$ —where j is a ballot identifier in I_B.

   "Destroy ballot j"

   $B := B \setminus \{B_j\}$ and $R := R \circ (\textsc{Destroy}, j, \mathtt{ub})$.

7. $(\textsc{SecretPermute}, i, b, j_0, j_1)$ —where i is the active player, $b \in \{0, 1\}$, and $j_0, j_1 \in I_B$ are distinct identifiers of ballots with the same level L.

   "Let Player i secretly permute ballots j_0 and j_1 (according to b)."

   $B := B \cup \{(\mathtt{ub} + 1, cont_B(j_b), L), (\mathtt{ub} + 2, cont_B(j_{1-b}), L)\}$; $B := B \setminus \{B_{j_0}, B_{j_1}\}$;
   $H_i := H_i \circ b$; $\mathtt{ub} := \mathtt{ub} + 2$; and $R := R \circ (\textsc{SecretPermute}, i, j_0, j_1, \mathtt{ub})$.

8. $(\textsc{BallotBox}, j_1, \ldots, j_k)$ —where $k \leq 5$ and $j_1, \ldots j_k \in I_B$ are distinct identifiers of ballots with the same level L.

   "Ballotbox $j_1, \ldots, j_k$"

   $p \leftarrow rand(\mathbb{S}_k)$; $B := B \cup \{(\mathtt{ub} + p(1), cont_B(j_1), L), \ldots, (\mathtt{ub} + p(k), cont_B(j_k), L)\}$;
   $B := B \setminus \{B_{j_1}, \ldots, B_{j_k}\}$; $\mathtt{ub} := \mathtt{ub} + k$; and $R := R \circ (\textsc{BallotBox}, j_1, \ldots, j_k, \mathtt{ub})$.

[23] All the ballot-box actions involving multiple super-envelopes require as inputs and produce as outputs the ballots of the same level (see below). Thus, the level of any ballot can be deduced from the public record.

We refer to a member of the first 6 classes as a *public action;* to a member of the 7th class as a *private action;* and to a member of the 8th class as an *action of Nature.* If $a = (\textsc{SecretPermute}, i, b, j_0, j_1)$, we refer to b as a's *hidden bit.* We say that private actions a and a' are *complementary* if $a = (\textsc{SecretPermute}, i, b, j_0, j_1)$ and $a' = (\textsc{SecretPermute}, i, 1-b, j_0, j_1)$; and refer to a as the *0-action* (of the pair) if $b = 0$, and as the *1-action* otherwise.

REMARKS.

- All ballot-box actions are deterministic functions, except for the actions of Nature.
- The variable `ub` never decreases and coincides with the maximum of all identifiers "ever in existence." Notice that we never re-use the identifier of a ballot that has left, temporarily or for ever, the table. This ensures that different ballots get different identifiers.
- Even though we could define the operations NewSup, PublicPermute, and BallotBox to handle an arbitrary number of ballots, it is a strength of our construction that we never need to operate on more than 5 ballots at a time. We thus find it convenient to define such bounded operations to highlight the practical implementability of our construction.

Definition 6: A global memory gm is *feasible* if there exists a sequence of global memories $gm^0, gm^1, \ldots, gm^k$, such that gm^0 is the empty global memory; $gm^k = gm$; and, for all $i \in [1, k]$, $gm^i = a^i(gm^{i-1})$ for some $a^i \in \mathcal{A}_{gm^{i-1}}$.

If (B, R, H) is a feasible memory, we refer to R as a *feasible public record.*

REMARK. If $gm = (B, R, H)$ is feasible, then $\mathcal{A}_{gm}$ is easily computable from R alone. Indeed, what ballots are in play, which ballots are envelopes and which are super-envelopes, *et cetera,* are all deducible from R. Therefore, different feasible global memories that have the same public record also have the same set of available actions. This motivates the following definition.

Definition 7: If R is a feasible public record, by $\mathcal{A}_R$ we denote the set of available actions for any feasible global memory with public record R.

Definition 8: An L-bit *ballot-box mechanism* is a tuple $\mathcal{B} = (N, K, J, PS, AF, OF)$ where

- N is a set of players;
- K, the *mechanism length,* is a positive integer;
- J, the *outcome-boundary round,* is a positive integer $\leq K$;
- PS, the *player sequence,* is a sequence of K elements from $N \cup \{\text{Nature}\}$: $PS = PS^1, \ldots, PS^K$;
- AF, the *action-function sequence,* is a sequence of K deterministic functions mapping public records to sets of ballot-box actions such that, for all $k \in [1, K]$ and for all feasible public record R:
  1. $AF^k(R) \in \mathcal{A}_R$ consists of either a single action, or two complementary private actions; and
  2. whenever $k > J$, $AF^k(R)$ is an open-envelope action.
- OF, the *outcome function,* is a function mapping Σ^* to $\{0, 1\}^L$.

Mechanism Play. Mechanism $\mathcal{B}$ is played as follows. Let $gm^0 = (B^0, R^0, H^0)$ be the empty global memory. The players and Nature act one at a time, in K rounds, as specified by PS: the first one to act is PS^1, the second is PS^2, and so on. At the kth round, if $j = PS^k \in N$, the information available to j consists of R^{k-1} and H_j^{k-1} —i.e., the current public record and the private history of j (i.e., the hidden bits of the secret actions played so far by j).[24] Based on this information, player j chooses an action $a^k \in AF^{k-1}(R^{k-1})$. Otherwise, if PS^k =Nature, then $AF^{k-1}(R^{k-1}) = \{a^k\}$, where a^k is an action of Nature. In either case, a^k maps the current global memory $gm^{k-1} = (B^{k-1}, R^{k-1}, H^{k-1})$ into a new global memory $a^k(gm^{k-1}) = gm^k = (B^k, R^k, H^k)$. After K rounds —i.e, when the global memory $gm^K = (B^K, R^K, H^K)$ is generated— the play terminates and the recommended outcome $y \in \{0,1\}^L$ is obtained by evaluating function OF on $Cont(R^{(J,K]})$ —i.e., the contents of the envelopes publicly opened in the last $K - J$ rounds.[25]

Executions. An execution e of $\mathcal{B}$ is a sequence of global memories obtainable by playing $\mathcal{B}$ until termination, $e = (B^0, R^0, H^0), \ldots, (B^K, R^K, H^K)$. By $B^k(e), R^k(e)$, and $H^k(e)$ we denote, respectively, the ballot set, the public record, and the private history profile of e at round k. The outcome of e, $out(e)$, is the string obtained by evaluating $OF(Cont(R^{(J,K]}))$.

Decision Nodes. A (round-k) decision node of player i in a mechanism $\mathcal{B}$ is a triple (k, R, H_i), where $k < K$ and $PS^k = i$, such that, for some execution e of $\mathcal{B}$, both $R = R^{k-1}(e)$ and $H_i = H_i^{k-1}(e)$. The set of all possible round-k decision nodes of player i (over all possible executions of $\mathcal{B}$) is denoted by DN_i^k.

Strategies. A (pure) strategy of player i in $\mathcal{B}$ is a function s_i mapping each possible decision node (k, R, H_i) of player i into the available action set $AF^k(R)$. We denote by $S_i^{\mathcal{B}}$ the set of all strategies of player i in $\mathcal{B}$, and by $S^{\mathcal{B}}$ the set of all strategy profiles in $\mathcal{B}$. We say that an execution e of $\mathcal{B}$ is an execution of $s \in S^{\mathcal{B}}$ if, for all $k \le K$, $a^k = s_i(k, R^{k-1}, H_i^{k-1})$ where $i = PS^k$. We denote by $EX(s)$ the distribution over the executions of s.[26] We may also write $EX_{\mathcal{B}}(s)$ to highlight, if needed, the mechanism $\mathcal{B}$.

REMARKS.

- *Dummy Players for Public Actions.* In a ballot-box mechanism, each public action is performed by the player specified in PS. However, because the effect of a public action solely depends on the current public record —not on who performs it— ballot-box mechanisms might as well specify that the first player performs all public actions. Ballot-box mechanisms may even "out-source" public actions to an external entity, such as a *referee*.
- *Finiteness.* Although there are infinitely many ballot-box actions, each ballot-box mechanism $\mathcal{B}$ is finite, because it consists of a finite number of rounds and at each round the active player has at most two actions to choose from.
- *Imperfect Information.* Ballot-box mechanisms are of imperfect-information for two reasons. First, whenever the set of available actions consists of two complementary secret actions, the active player permutes a pair of ballots, but all other players do not know which of two

[24]This implies that j knows all the actions he played in the past. In fact, all public actions are manifest in the public record, and j's secret actions can be fully reconstructed from their hidden bits and the public record.

[25]Recall that the outcome-boundary round J is defined so that, for each $k = J+1$ to K, $R^k = (\text{OpenEn}, j^k, c^k)$. Thus, $Cont(R^{(J,K]}) = c^{J+1} \cdots c^K$.

[26]Because all strategies of s are pure (and therefore deterministic), this distribution solely arises from the randomness of the ballot box (if used).

possible permutations was actually chosen. Second, when called into action, the ballot box (i.e., Nature) permutes a sequence of ballots in a way unknown to all players. For these two reasons, the players may not know the exact content of a particular envelope. But they do know (1) which envelopes are in play —because the identifiers of the current ballots are always deducible from the public record— and (2) the exact *set* of the current envelopes' contents —because the content of each newly created envelope is always public, and remains unaltered until the envelope is opened.

- *Extended Ballot-Box mechanisms.* One can easily envisage extending the above definition of a ballot-box mechanism to allow for richer action sets, more general rules to select the acting players, and more complex information structures. This might be quite desirable for achieving some other objectives not considered in this paper. We chose to have a bare-bone definition to keep the notation as simple as possible and to capture required elements for our perfect implementation exactly.

D. THE NOTION OF A PERFECT IMPLEMENTATION

D.1. Preliminary Notions

D.1.1. Bit-by-bit Encodings

Let ℓ be a positive integer. A *bit-by-bit encoding* of length ℓ is a function mapping any binary string x to a string $\bar{x} \in \Sigma^*$ satisfying the following 3 properties: (1) $\bar{0}, \bar{1} \in \Sigma^\ell$; (2) $\bar{0} \neq \bar{1}$; and (3) $\overline{xy} = \bar{x}\bar{y}$ (i.e., it preserves concatenation). Because a bit-by-bit encoding is fully specified by the values it takes at 0 and 1, we identify it with the pair of strings $(\bar{0}, \bar{1})$.

A bit-by-bit encoding $(\bar{0}, \bar{1})$ is immediately extended to binary functions. If $f : D \to \{0,1\}^*$, where $D \subset \{0,1\}^*$, then we define the function $\bar{f}$ as follows: $\bar{f}(\bar{x}) = \overline{f(x)}$ for all $x \in D$.

The *decoding function of* $(\bar{0}, \bar{1})$ is the function mapping $\bar{x}$ to x for all $x \in \{0,1\}^*$.

(As we shall see, the bit-by-bit encoding we use in our construction is an encoding of length 5, in which 0 and 1 are represented by two distinct permutations of the first five integers.)

D.1.2. Ballot-Box Protocols

DEFINITION 9: A *ballot-box protocol* $\mathcal{P}$ is a ballot-box mechanism which has its length equal to its outcome-boundary round —i.e., $\mathcal{P} = (N, K, K, PS, AF, OF)$— and is executed starting with any feasible global memory.

For short, when no confusion is possible, we may just use the term *protocol* rather than "ballot-box protocol". The distribution over the executions of a protocol $\mathcal{P}$ with strategy profile s and initial global memory gm^0 is denoted by $EX_{\mathcal{P}}(s, gm^0)$.

REMARKS.

- Being always equal to the length K, the outcome-boundary round of a ballot-box protocol $\mathcal{P} = (N, K, K, PS, AF, OF)$ is redundant; and so is $\mathcal{P}$'s outcome function OF, because it is to be evaluated on the last $K - K = 0$ elements of the public record. Accordingly, we identify $\mathcal{P}$ with its 1st, 2nd, 4th, and 5th components and more simply write $\mathcal{P} = (N, K, PS, AF)$.
- As "degenerate" ballot-box mechanisms, protocols retain all notions (e.g., executions, strategies, and decision nodes) and notations (e.g., $B^k(e), R^k(e)$, and $H^k(e)$) of Section C.

DEFINITION 10: We say that a ballot-box protocol $\mathcal{P} = (N, K, PS, AF)$ is the *concatenation* of protocols $\mathcal{P}_1 = (N, K_1, PS_1, AF_1), \ldots, \mathcal{P}_m = (N, K_m, PS_m, AF_m)$ if

- $K = K_1 + \cdots + K_m$;
- $PS = PS_1 \circ \cdots \circ PS_m$; and
- $AF = AF_1 \circ \cdots \circ AF_m$.

We denote the concatenation of protocols $\mathcal{P}_1, \ldots, \mathcal{P}_m$ by $\mathcal{P} = \mathcal{P}_1 \circ \cdots \circ \mathcal{P}_m$.

D.1.3. Address Vectors

DEFINITION 11: An *address* is a finite sequence x of distinct positive integers. An *address vector* x is a vector of mutually disjoint addresses, that is, $\{x_i\} \cap \{x_j\} = \phi$ whenever $i \neq j$. The *identifier set* of an address vector $x = (x_1, \ldots, x_k)$ is denoted by I_x and defined to be the set $\bigcup_{i=1}^{k} \{x_i\}$. If B is a set of ballots, then we define $cont_B(x)$ to be the vector $(cont_B(x_1), \ldots, cont_B(x_k))$. If i is a positive integer, then $x + i$ is the address vector whose jth component is $x_j + i$ (i.e., each element of sequence x_j is increased by i).

As usual, an address profile is an address vector indexed by the set of players.

D.2. Ballot-Box Committers

As intuitively explained in Section 3, a perfect implementation of a normal-form mechanism $\mathcal{M}$ consists of 3 protocols: a committer, a computer, and a revealer. The committer is executed first. At the highest level, its goal is to implement the stage of $\mathcal{M}$ in which the players' independently choose their private messages. A ballot-box committer achieves its goal by generating, for each player i, a separate sequence of envelopes whose contents encode his message m_i. (Such messages are thus "committed" in the sense that the players cannot change them, because the envelopes containing them are in public view.) To match $\mathcal{M}$'s first stage exactly, such envelopes must be generated in a special way, that is, so as to guarantee the three properties formally described below.

DEFINITION 12: Let $\mathcal{P} = (N, K, SP, AF)$ be a ballot-box protocol, $(\bar{0}, \bar{1})$ a bit-by-bit encoding, and x an address profile, $x = x_1, \ldots, x_n$. We say that $\mathcal{P}$ is a L-bit *ballot-box committer* for $(\bar{0}, \bar{1})$ with output address profile x if there exists a unique sequence U such that, for every execution e of $\mathcal{P}$ whose initial global memory is empty, the following three properties hold:

1. *Correctness:* For every player i, $H_i^K(e) \in \{0,1\}^L$ and $cont_{B^K(e)}(x_i) = \overline{H_i^K(e)}$.
2. *Privacy:* $R^K(e) = U$.
3. *Clean Termination:* $I_{B^K(e)} = I_x$.

   We denote the *final identifier upper-bound* of an execution of $\mathcal{P}$ by $\texttt{ub}_{\mathcal{P}}$.

REMARKS.

- *Correctness.* A committer may involve several types of ballot-box operations. (Indeed, ours generates both new envelopes and new super-envelopes, some of which it opens and some of which it destroys.) No matter which operations these may be, however, Correctness requires that, once $\mathcal{P}$ is executed, the private history string of each player is of length L, and that the ballots corresponding to the output address profile $x_1, \ldots, x_n$ contain the bit-by-bit encoding of the players' hidden-bit sequences. This requirement has both syntactic and semantic implications. Syntactically, Correctness implies that each element of x_i is the identifier of an envelope in $B^K(e)$ —else, $cont_{B^K(e)}(x_i)$ would equal $\perp$ rather than the bit-by-bit encoding of an L-bit string. Further, because each envelope of an output address contains a single symbol of Σ, each x_i consists of ℓL identifiers whenever the bit-by-bit encoding scheme $(\bar{0}, \bar{1})$ has length ℓ. Semantically, Correctness implies that the envelopes of address x_i encode a message m_i freely chosen by player i alone. (I.e., the other players have no control over the value of m_i.) This is so because the envelopes in x_i are guaranteed to contain the bit-by-bit encoding of the final private history of player i. Recall that i's private history, H_i, grows, by a bit at a time, in only one case: when i himself secretely chooses one of two complementary actions. In our physical interpretation, when i temporarily holds two envelopes behind his back, and then returns them in the order he chooses. This is an "atomic" (in the sense of "indivisible") choice of i, and therefore the other players have no control over it.

- *Privacy.* Privacy requires that, at each round k of a ballot-box committer, the public information available to the acting player always consists of the fixed sequence U^{k-1}, no matter what strategies the players employ. Thus, while Correctness implies that any control over the choice of a player's message solely rests with that player, Privacy implies that all messages are independently chosen, as demanded in a normal-form mechanism $\mathcal{M}$, and totally secret at the end of the committer's execution.[27] Privacy thus rules out any possibility of signaling among players during the execution of a committer.

  Note that the information available to a player i consists of both his private history H_i and the public record R. In principle, therefore, the privacy condition should guarantee that no information about other players' strategies is deducible from H_i and R *jointly.* As argued above, however, H_i^K depends on i's strategy alone. Thus, formulating Privacy in terms of the public record alone is sufficient.

- *Clean Termination.* Clean termination ensures that only the envelopes containing the desired encoding of the players' private messages remain on the table.

- *Fixed Final Identifier Upper-Bound.* Notice that the value $\mathtt{ub}_{\mathcal{P}}$ is well defined because a committer's execution starts with an empty global memory and ends with a fixed public record. Notice too that $\mathtt{ub}_{\mathcal{P}}$ is computable form U alone.

D.3. Ballot-Box Computers

A ballot-box computer $\mathcal{P}$ for a function f is a special protocol. Executed on an initial global memory in which specific envelopes (the "input envelopes") contain an input x for f, $\mathcal{P}$ replaces such envelopes with new ones (the "output envelopes") that will contain the corresponding output $f(x)$. Of course, no property is required from $\mathcal{P}$ if its initial memory is not of the proper form.

With modularity in mind, we actually envision that an execution of a computer $\mathcal{P}$ may be preceded and/or followed by the execution of other computers. We thus insist that $\mathcal{P}$ does not

[27]Traditionally, such independence property is also referred to as *simultaneity.* More strongly, one might interpret simultaneity to mean that the players of $\mathcal{M}$ literally submit their privately chosen messages at the *same physical time.* As currently defined, a perfect implementation of $\mathcal{M}$ does not preserve this property. (Indeed, when implementing committers in Section F.3, the players commit their messages sequentially, and one bit at a time.) If literal simultaneity were indeed possible, however, we could certainly incorporate it into ballot-box mechanisms and demand that the players execute each step of the commitment phase simultaneously.

"touch" any ballots of the initial memory besides its input envelopes. This way, partial results already computed, if any, will remain intact.

Definition 13: Let $f : X^a \to Y^b$ be a finite function, where $X, Y \subset \Sigma^*$; and let $x = x_1, \ldots, x_a$ be an address vector. We say that a feasible global memory $gm = (B, R, H)$ is *proper* for f and x if $I_x \subset I_B$ and $cont_B(x) \in X^a$.

Definition 14: Let $f : X^a \to Y^b$ be a finite function, where $X, Y \subset \Sigma^*$; let x and y be two address vectors; and let $\mathcal{P} = (N, K, PS, AF)$ be a ballot-box protocol such that AF only returns sets consisting of a single action: either a public one, or an action of Nature. We say that $\mathcal{P}$ is a *ballot-box computer for f* with input address vector x and output address vector y if there exists a probabilistic algorithm SIM such that, for any execution of $\mathcal{P}$ with (i) strategy profile s; and (ii) initial memory $gm^0 = (B^0, R^0, H^0)$, proper for f and x, whose identifier upper-bound is $\mathtt{ub}$, the following three properties hold:

1. *Correctness:* $\langle\ cont_{B^K(e)}(y + \mathtt{ub}) \ :\ e \leftarrow EX_{\mathcal{P}}(s, gm^0)\ \rangle \ =\ f(cont_{B^0}(x))$.
2. *Privacy:* $\langle\ R^K(e) \ :\ e \leftarrow EX_{\mathcal{P}}(s, gm^0)\ \rangle = \langle\ R^0 \circ F \ :\ F \leftarrow SIM(\mathtt{ub})\ \rangle$.
3. *Clean Operation:* $B^K(e) = B_{\{y+\mathtt{ub}\}} \cup B^0 \setminus B_{\{x\}}$.

We refer to SIM as $\mathcal{P}$'s *simulator*; to $B_{\{x\}}$ as the *input envelopes*; and to $B_{\{y+\mathtt{ub}\}}$ as the *output envelopes*. For short, when no confusion may arise, we refer to $\mathcal{P}$ as a *computer*.

remarks.

- *Correctness.* Semantically, Correctness states that the output envelopes will contain f evaluated on the contents of the input envelopes. Syntactically, as for committers, Correctness implies that each integer of each address $y_j + \mathtt{ub}$ is the identifier of an envelope in $B^K(e)$.
- *Privacy.* By running a computer $\mathcal{P}$ for f, the only *additional* information about f's inputs or outputs gained by the players consists of $R^{[1,K]}$, the portion of the public record generated by $\mathcal{P}$'s execution. Privacy guarantees that this additional information is actually *useless.* Indeed, (1) the identifier upper-bound $\mathtt{ub}$ is deducible from R^0, and thus already known to the players, and (2) $R^{[1,K]}$ can be generated "with exactly the same odds as in a random execution of $\mathcal{P}$" by $\mathcal{P}$'s simulator only on input $\mathtt{ub}$. Since such simulator is a *fixed algorithm*, $R^{[1,K]}$ cannot contain any information about f's inputs and outputs that is not deducible from $\mathtt{ub}$.
- *Clean Operation.* Clean Operation guarantees that $\mathcal{P}$
  1. Never touches an initial ballot that is not an input envelope (in fact, if a ballot is acted upon, then it is either removed from the ballot set, or receives a new identifier), and
  2. Eventually replaces all input envelopes with the output envelopes (i.e., other ballots generated by $\mathcal{P}$ are temporary, and will not exist in the final ballot set).
- *Dummy Strategies.* Note that the strategies s formally enter our definition of a computer $\mathcal{P}$ (since they are part of our notation for a random execution) but are actually irrelevant, because all action sets of $\mathcal{P}$ have cardinality 1.[28]

[28]This constraint guarantees that players cannot signal to one another during the execution of $\mathcal{P}$. It also makes it further clear that, although the entire information available to a player i consists of the public record R as well as his own private record H_i, it suffices to state Privacy in term of the public record alone. In fact the private record H_i of each player i grows only when i chooses his action from an action set consisting of two complementary actions. In

- *No Bit-By-Bit Encodings.* Notice that, while the definition of a committer relies on bit-by-bit encodings, that of a computer does not. This is so because committers must prepare the inputs to a mechanism's outcome function, and such function maps bit strings to bit strings. Our computers, on the other hand, deal with functions with more general domains and ranges. For instance, one of our fundamental ballot-box computers must evaluate the function that, on input a permutation of 5 elements, returns the inverse permutation. Although also the inputs and outputs of such a function may ultimately be encoded as bit strings, we prefer to represent them as sequences of the first 5 integers. For instance, the identity permutation is represented by the sequence 12345. Indeed the ability of computing with ballots and a ballot box in a private and correct way crucially depends on the representation chosen for inputs and outputs, and as we shall see representing permutations of 5 elements as strings over the alphabet $\{1, 2, 3, 4, 5\}$ will be crucial.

D.4. Ballot-Box Revealers

A ballot-box revealer is a protocol that opens a fixed sequence of envelopes.

Definition 15: We say that a ballot-box protocol $\mathcal{P} = (N, K, PS, AF)$ is a *ballot-box revealer* with address vector z if, for any initial global memory (B^0, R^0, H^0) such that each member of I_z is the identifier of an envelope in B^0, (1) each function in AF only returns action sets consisting of a single open-envelope action; and (2) in any execution of $\mathcal{P}$, all envelopes in I_z are opened.

D.5. Perfect Implementation

A ballot-box mechanism $\mathcal{B}$ perfectly implementing a normal-form mechanism $\mathcal{M}$ simply consists of three ballot-box protocols, executed in order: (1) a committer that generates "input envelopes" whose contents encode the players's choices for $\mathcal{M}$'s messages; (2) a computer that evaluates the outcome function of $\mathcal{M}$ on the contents of the input envelopes and stores the result in "output envelopes"; and (3) a revealer that opens all such output envelopes, thus enabling the players —or anyone else— to retrieve the desired result. By the end (since in this paper we do not deal with executing sequences of mechanisms), no ballot should remain on the table.

Definition 16: Let $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$ be a standard normal-form mechanism; $(\bar{0}, \bar{1})$ a bit-by-bit encoding; $\mathcal{P}_1$ an L-bit committer for $(\bar{0}, \bar{1})$ with output address profile x and final identifier upper-bound $\texttt{ub}^*$; $\mathcal{P}_2$ a ballot-box computer for $\bar{g}$ with input address profile x and output address vector y; and $\mathcal{P}_3$ a ballot-box revealer with input address vector $y + \texttt{ub}^*$.

We say that a ballot-box mechanism $\mathcal{B} = (N, K, J, PS, AF, OF)$ *perfectly implements* $\mathcal{M}$ if

1. The ballot-box protocol (N, K, PS, AF) is the concatenation of $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$; and
2. OF is the decoding function of $(\bar{0}, \bar{1})$, applied to the contents of the envelopes opened by $\mathcal{P}_3$.

REMARKS.

any case, as argued for committers, the content H_i is not affected by any action of the other players, but grows of one bit only when i privately selects one of two complementary actions. Thus, if the content of H_i contained knowledge about the other players' strategies, this knowledge would have been available to i —through the public record!— when he chose his private actions.

- *No Signaling.* A distinctive property (and often the main advantage) of a mediated normal-form mechanism $\mathcal{M}$ is that the players cannot signal to each other.[29] This property, though not true for most extensive-form mechanisms, actually holds for any ballot-box mechanism $\mathcal{B}$ that perfectly implements $\mathcal{M}$. This is quickly argued as follows. Signalling essentially arises from a player's ability to choose among different actions. In $\mathcal{B}$, instead, players can choose between different actions only during the execution of the ballot-box committer. As already argued, however, all executions of a ballot-box committer produce exactly the same public record, regardless of their chosen strategies, and thus no player can signal to any other.
- *No Residual Information.* Note that our definition of perfect implementation implies that, by $\mathcal{B}$'s end, "no information is left on the table." (In fact, $\mathcal{P}_1$'s Clean Termination implies that only the envelopes whose contents encode the players' messages remain on the table at the end of the committal stage; $\mathcal{P}_2$'s Clean Operation implies that only the output envelopes remain on the table at the end of the computation stage; and $\mathcal{P}_3$ ensures that no more ballots are finally left on the table.) This property is very important, because it is hard —if not impossible!— to constrain what may happen after a mechanism ends, and thus to control what happens to any residual information in the system. Indeed, if such information existed and were related to the players' strategies, its fate might affect the very play of the mechanism. For instance, let $\mathcal{M}$ be the following normal-form mechanism for running a secret referendum: player i's message m_i is either YES or NO, and the public outcome is the tally of YES votes. Let now $\mathcal{M}'$ be an implementation of $\mathcal{M}$ that, while correctly and privately computing the desired tally, also generates and hands to each player i a sequence of unopened envelopes, E_i, encoding m_i. Such an $\mathcal{M}'$ can hardly be considered to be a satisfactory implementation of $\mathcal{M}$, as it makes "buying votes" easier than in $\mathcal{M}$. Indeed, in $\mathcal{M}'$ i can ex-post prove how he voted to some other player j by handing over the envelopes of E_i to j, or by opening them in front of him. By contrast, after a play of $\mathcal{M}$ player i could only *claim* to another player j what his vote m_i was. (In a forthcoming paper dealing with multiple mechanisms, Property 3 will be properly revisited.)

E. PERFECT IMPLEMENTATION IMPLIES STRATEGIC AND PRIVACY EQUIVALENCE

Let us now prove that the structural definition of a perfect implementation actually captures, as promised, the intuitively discussed properties of strategic and privacy equivalence. We start by formalizing them.

DEFINITION 17: Consider a ballot-box mechanism $\mathcal{B} = (N, K, J, PS, AF, Y, OF)$ and a standard normal-form mechanism $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$. We say that $\mathcal{B}$ is *strategic and privacy equivalent* to $\mathcal{M}$ if $\forall i \in N$ there exist a bijection $\psi_i : S_i^{\mathcal{B}} \leftrightarrow \{0,1\}^L$ and a probabilistic algorithm SIM such that, $\forall s \in S^{\mathcal{B}}$ the following properties hold:

Strategic Equivalence: $\langle\, out(e) \,:\, e \leftarrow EX_{\mathcal{B}}(s) \,\rangle \;=\; g(\psi_1(s_1), \ldots, \psi_n(s_n))$.

Privacy Equivalence: $\langle\, R^K(e) \,:\, e \leftarrow EX_{\mathcal{B}}(s) \,\rangle = \langle\, F \,:\, e \leftarrow EX_{\mathcal{B}}(s);\; F \leftarrow SIM(out(e)) \,\rangle$.

We refer to algorithm SIM as $\mathcal{B}$'s *simulator*.

[29]Of course, players may always "communicate to one another via the outcome" —at least for most outcome functions. That is, in $\mathcal{M}$, by sending the mediator a suitable message m_i, a player may affect the outcome so as communicate some information to the other players. Such a communication is both "legitimate and intrinsic." By saying that a player cannot signal we mean that he cannot communicate more information to the other players than via the outcome alone.

THEOREM 2: *If a ballot-box mechanism $\mathcal{B}$ perfectly implements a standard normal-form mechanism $\mathcal{M}$, then $\mathcal{B}$ is strategically and privacy equivalent to $\mathcal{M}$.*

Proof. Let $\mathcal{B} = (N, J, K, PS, AF, OF)$ be the concatenation of a committer $\mathcal{P}_1$ of length K_1, a computer $\mathcal{P}_2$ of length K_2, and a revealer $\mathcal{P}_3$ of length K_3; let $(\bar{0}, \bar{1})$ be the bit-by-bit encoding scheme of $\mathcal{B}$; and let ℓ be the length of $(\bar{0}, \bar{1})$.

PROOF OF STRATEGIC EQUIVALENCE. First of all, notice that a strategy of a player i in $\mathcal{B}$ is uniquely identified with his strategy in $\mathcal{P}_1$. (This is so, because every action set of $\mathcal{P}_2$ and $\mathcal{P}_3$ has cardinality 1.)

Let us now define, for each player i, the function ψ_i as follows. On input $s_i \in S_i^{\mathcal{B}}$, do: choose any strategy sub-profile $s_{-i}^* \in S_{-i}^{\mathcal{B}}$; choose any execution e of $\mathcal{B}$ with strategy profile (s_i, s_{-i}^*); and set $\psi_i(s_i) = H_i^{K_1}(e)$. That is, define $\psi_i(s_i)$ to be the hidden-bit sequence of i at the end of $\mathcal{P}_1$. Function ψ_i is well defined, because in the execution of a committer protocol a player's hidden-bit sequence solely depends on that player's strategy. Furthermore, because $\mathcal{P}_1$ is an L-bit committer, $H_i^{K_1}(e)$ is an L-bit string. Thus, $\psi_i : S_i^{\mathcal{B}} \to \{0,1\}^L$.

Let us now argue that ψ_i is surjective. Let $b_i = b_i^1 \dots b_i^L \in \{0,1\}^L$, and let $s_i \in S_i^{\mathcal{B}}$ be the strategy that selects the b_k-action at the kth decision node of i where the action set consists of two complementary private actions. Then, $\psi_i(s_i) = b_i^1 \cdots b_i^L$. Let us now argue that ψ_i is injective. Assume that s_i and s_i' are distinct strategies in $S_i^{\mathcal{B}}$. Then, because $\mathcal{P}_2$'s and $\mathcal{P}_3$'s action sets always have cardinality 1, s_i and s_i' must prescribe different actions in at least one of the L decision nodes of i in P_1 in which the action set consists of two complementary actions. If they do so at the jth such node, then $\psi_i(s_i)$ and $\psi_i(s_i')$ would differ at the jth bit. Thus, ψ_i is a bijection: $\psi_i : S_i^{\mathcal{B}} \leftrightarrow \{0,1\}^L$.

Finally, let $e = gm^0, \dots, gm^K$ be a random execution of $\mathcal{B}$ with strategy profile s, where $gm^j = (B^j, R^j, H^j)$. Then, the first K_1 global memories of e constitute an execution of committer $\mathcal{P}_1$; the next K_2 global memories constitute an execution of computer $\mathcal{P}_2$ on initial global memory gm^{K_1}; and the final K_3 global memories constitute an execution of revealer $\mathcal{P}_3$ on initial global memory $gm^{K_1+K_2}$. The correctness property of $\mathcal{P}_1$ implies that the content of its jth output address is $\overline{H_j^{K_j}}$. Because the input addresses of $\mathcal{P}_2$ equal the output addresses of $\mathcal{P}_1$, because $\mathcal{P}_2$ is a computer for $\bar{g}$, and because e is a random execution of $\mathcal{P}_2$ on initial global memory gm^{K_1}, by the correctness property of $\mathcal{P}_2$, the contents of the output addresses of $\mathcal{P}_2$ are distributed according to $\overline{g(H_1^{K_1}, \dots, H_i^{K_1})}$. Since $\mathcal{P}_3$ is a revealer whose input addresses equal the output addresses of $\mathcal{P}_2$, its length K_3 is equal to ℓL, and, upon termination of $\mathcal{P}_3$, the outcome returned by OF is distributed according to $g(H_1^{K_1}, \dots, H_i^{K_1})$. Therefore, for any strategy profile $s \in S^{\mathcal{B}}$, $\langle\, out(e) : e \leftarrow EX_{\mathcal{B}}(s) \,\rangle$ and $g(\psi_1(s_1), \dots, \psi_n(s_n))$ are equal distributions.

PROOF OF PRIVACY EQUIVALENCE. Consider the following algorithm SIM, in which U is the fixed public record of $\mathcal{P}_1$, SIM_2 is the simulator for $\mathcal{P}_2$, y is the output address vector of $\mathcal{P}_2$, and the encoding scheme is the same bit-by-bit encoding of $\mathcal{B}$.

Algorithm *SIM:*

```
Input: $z = z_1 \cdots z_L \in \{0,1\}^L$.
(1) $F_2 \leftarrow SIM_2(\mathtt{ub}_{\mathcal{P}_1})$;
(2) Compute $\bar{z} = b_1 \cdots b_{\ell L}$;
    (Note that $\bar{z} \in \{0,1\}^{\ell L}$, because $(\bar{0}, \bar{1})$ has length $\ell$, and that $y = y_1, \dots, y_{\ell L}$.)
```

(3) Set $F_3 = (\textsc{OpenEn}, y_1 + \mathtt{ub}_{\mathcal{P}_1}, b_1), \ldots, (\textsc{OpenEn}, y_{\ell L} + \mathtt{ub}_{\mathcal{P}_1}, b_{\ell L})$.
(I.e., F_3 is the public record produced by opening an envelope sequence $y + \mathtt{ub}_{\mathcal{P}_1}$ containing $\bar{z}$.)

Output: Public record $F = U \circ F_2 \circ F_3$.

It is clear that SIM is the simulator required for $\mathcal{B}$'s privacy equivalence to $\mathcal{M}$ to hold.

Q.E.D.

F. HOW TO PERFECTLY IMPLEMENT ANY STANDARD NORMAL-FORM MECHANISM

As per Definition 16, to construct a perfect ballot-box implementation of a normal-form mechanism, we must first specify a bit-by-bit encoding scheme and then (1) a ballot-box committer; (2) a ballot-box computer for $\bar{g}$; and (3) a ballot-box revealer.

F.1. Our Encoding Scheme

In this section we present our encoding scheme, respectively mapping 0 and 1 to two distinct permutations in $\mathbb{S}_5$, that is, to two specific bijections from $\{1, 2, 3, 4, 5\}$ into itself. The reason for this unusual choice of encodings is that, using our ballot-box operations, we can perform arbitrary computation on "enveloped data."

F.1.1. Barrington's $\mathbb{S}_5$ Implemention of NOTs and ANDs

The following is our rendition of a result of Barrington (1986), namely that the Boolean functions NOT and AND can be realized as sequences of group operations in $\mathbb{S}_5$.

NOTATION. If x is a permutation in $\mathbb{S}_5$, we denote by x_j the image of integer $j \in \{1, 2, 3, 4, 5\}$ under x —i.e., $x_j = x(j)$— and identify x with the 5-symbol string $x_1x_2x_3x_4x_5$.

SIX CONSTANTS. There exist six permutations in $\mathbb{S}_5$, denoted by $\mathcal{I}, a, b, [a \to b], [a^{-1} \to a]$, and $[aba^{-1}b^{-1} \to a]$ which satisfy the following properties:

- $\mathcal{I}$ is the identity permutation.
- $aba^{-1}b^{-1} \neq \mathcal{I}$.
- $[a \to b]^{-1} a [a \to b] = b$.
- $[a^{-1} \to a]^{-1} a^{-1} [a^{-1} \to a] = a$.
- $[aba^{-1}b^{-1} \to a]^{-1} aba^{-1}b^{-1} [aba^{-1}b^{-1} \to a] = a$.

Proof: $a = 12453$, $b = 25341$, $[a \to b] = 34125$, $[a^{-1} \to a] = 12354$, and $[aba^{-1}b^{-1} \to a] = 42153$.

THREE OPERATORS. For any $x \in \mathbb{S}_5$, the operators " $\tilde{}$ " , " $'$ " and " $*$ " are defined as follows:

- $\tilde{x} = [a \to b]^{-1} x [a \to b]$
- $x' = [aba^{-1}b^{-1} \to a]^{-1} x [aba^{-1}b^{-1} \to a]$
- $x^* = [a^{-1} \to a]^{-1} x [a^{-1} \to a]$

BIT REPRESENTATION. 0 is represented by $\mathcal{I}$, and 1 by $a = 12453$.

REALIZING NOT & AND. If permutations x_1 and x_2 respectively represent bits b_1 and b_2 then

- $\neg b_1 = (x_1 a^{-1})^*$
- $b_1 \wedge b_2 = (x_1 \tilde{x_2} x_1^{-1} \tilde{x_2}^{-1})'$

F.1.2. Insufficiency of NOTs and ANDs for General Computation

Unfortunately, NOT and AND are insufficient for general computation.[30] To implement any deterministic finite function, in addition to NOT and AND, one needs to use also the binary function $DUPLICATE$, which on input a bit b returns the pair (b, b). And to implement any *probabilistic* finite function, one further needs to use the binary function $COIN$, which on empty input returns a random bit. However, neither $DUPLICATE$ nor $COIN$ can be realized in $\mathbb{S}_5$.

At high level, we bypass this limitation by (1) representing permutations in $\mathbb{S}_5$ as sequences of 5 envelopes and (2) using these *physical* representations and our ballot-box operations for implementing $DUPLICATE$ and $COIN$ (in addition to NOT and AND). That is, rather than viewing a permutation in $\mathbb{S}_5$ as a single, 5-symbol string, we view it a sequence of 5 distinct symbols, and put each one of them into its own envelope, which can then be manipulated separately by our ballot-box operations. Such "segregation" of permutations of $\mathbb{S}_5$ into separate envelopes is crucial to our ability of performing general computation, and in a private way too.

F.1.3. Our Envelope Encodings

Definition 18: We define the $\mathbb{S}_5$ *encoding* to be the following, length-5, bit-by-bit encoding:
$(\bar{0}, \bar{1}) = (\mathcal{I}, a)$, where $\mathcal{I} = 12345$ and $a = 12453$.

Definition 19: Let B be a well-defined set of ballots and σ a sequence of 5 envelope identifiers in I_B. If $cont_B(\sigma) \in \mathbb{S}_5$, then we say that σ is an *envelope encoding* of a permutation of $\mathbb{S}_5$ and refer to such a permutation by $\hat{\sigma}$. If $\hat{\sigma} \in \{\mathcal{I}, a\}$, then σ is an *envelope encoding of a bit.*

Therefore, tautologically, an envelope encoding σ is the envelope encoding of $\hat{\sigma}$. We reserve lower-case Greek letters to denote envelope encodings.

F.2. Our Protocol Conventions

We specify our protocols using the following conventions.

1. For simplicity, we directly list the action(s) available at a given round, rather than providing an algorithmic description of the action function AF operating on the public record. If a listed action a is not feasible for the current memory, we simply interpret it as 'Make a new envelope with public content 0," which is always feasible.
2. We describe an individual action a via its "informal reference" as per Definition 5. In addition, we add an explicit and convenient reference to the identifier(s) of any ballot generated by a. For instance, when we say "Make a new envelope x with public content c", we mean "Make a new envelope with public content c" and refer to the newly created identifier as x (rather than $\texttt{ub} + 1$).
3. If a is a public action or an action of Nature, then we omit specifying the acting player, since the action's effect will be the same no matter who performs it.

[30] Indeed, the result of Barrington (1986) was solely concerned with implementing a restricted class of finite functions called NC^1. Notice that, when computing the AND of two bits in $\mathbb{S}_5$, the method above destroys both of them. A general computation may instead require to perform additional operations on such bits, and thus one needs to be able of "saving copies of them."

4. We often collapse the actions of several rounds into a single *conceptual round,* providing convenient names for the ballot identifiers generated in the process. For instance, if p is a permutation in $\mathbb{S}_5$, the conceptual round "Create an envelope encoding σ of p" stands for the following 5 actions:

   Make a new envelope σ_1 with public content p_1.
   Make a new envelope σ_2 with public content p_2.
   Make a new envelope σ_3 with public content p_3.
   Make a new envelope σ_4 with public content p_4.
   Make a new envelope σ_5 with public content p_5.

F.3. Our Ballot-Box Committers

Protocol $Commit_L$

For player $i = 1$ to n DO: For bit $t = 1$ to L Do:
(1) Create an envelope encoding $\alpha^{(i,t)}$ of permutation $\mathcal{I} = 12345$.
(2) Create an envelope encoding $\beta^{(i,t)}$ of permutation $a = 12453$.
(3) Make a new super-envelope $A^{(i,t)}$ containing envelopes $\alpha_1^{(i,t)}, \ldots, \alpha_5^{(i,t)}$.
(4) Make a new super envelope $B^{(i,t)}$ containing envelopes $\beta_1^{(i,t)}, \ldots, \beta_5^{(i,t)}$.
(5) Let player i secretly permute $A^{(i,t)}$ and $B^{(i,t)}$ to obtain the super-envelopes $C^{(i,t)}$ and $D^{(i,t)}$.
(6) Open $C^{(i,t)}$ to expose envelopes $\gamma_1^{(i,t)}, \ldots, \gamma_5^{(i,t)}$. Set $\gamma^{(i,t)} = \gamma_1^{(i,t)}, \ldots, \gamma_5^{(i,t)}$.
(7) Destroy $D^{(i,t)}$.
Output Addresses: For each $i \in N$, the sequence $x_i = \gamma^{(i,1)}, \ldots, \gamma^{(i,L)}$.

LEMMA 1: *Protocol $Commit_L$ is an L-bit Ballot-Box Committer for the $\mathbb{S}_5$ encoding.*

PROOF OF CORRECTNESS. At the end of Step 3, $A^{(i,t)}$ contains a sequence of 5 envelopes encoding the identity permutation $\mathcal{I}$, and, at the end of Step 4, $B^{(i,t)}$ contains a sequence of 5 envelopes encoding permutation a. Thus, recalling that in the $\mathbb{S}_5$ encoding $\mathcal{I} = \bar{0}$ and $a = \bar{1}$, at the end of Step 5, $C^{(i,t)}$ contains an envelope encoding of $\bar{b}$ if player i "chooses the bit b" (i.e., if the tth bit of H_i is b). Thus, at the end of Step 6, the content of address x_i is $\overline{H_i}$, where H_i is a binary string of length L, as demanded by Definition 10.

PROOF OF PRIVACY. First notice that, at each iteration (i,t) of the seven steps of $Commit_L$, the variable `ub` is increased by 19. (In fact: Steps 1 and 2 consist of five actions, each action producing one new identifier; Steps 3 and 4 produce one new identifier each; Step 5 produces 2 new identifiers; Step 6 opens one super-envelope, thus exposing five envelopes and producing five new identifiers; finally, Step 7 destroys one super-envelope without generating any new identifiers.) Now notice that, prior to iteration (i,t), exactly $(i-1)L+(t-1)$ iterations have occurred. Therefore, the value of `ub` at the beginning of iteration (i,t) is $19((i-1)L+(t-1))$. Thus (recalling that $a = 12453$), Privacy is established by noticing that, in any execution of $Commit_L$ —no matter what the bits committed so far may be!— the portion of the public record generated by iteration (i,t) is the following, *fixed* sequence $U(i,t)$, where ub is the integer $19((i-1)L+(t-1))$:

$(\text{NewEn}, 1, ub+1)$ $(\text{NewEn}, 2, ub+2)$ $(\text{NewEn}, 3, ub+3)$ $(\text{NewEn}, 4, ub+4)$ $(\text{NewEn}, 5, ub+5)$
$(\text{NewEn}, 1, ub+6)$ $(\text{NewEn}, 2, ub+7)$ $(\text{NewEn}, 4, ub+8)$ $(\text{NewEn}, 5, ub+9)$ $(\text{NewEn}, 3, ub+10)$

(NewSup, $ub+1, \ldots, ub+5, ub+11$)
(NewSup, $ub+6, \ldots, ub+10, ub+12$)
(SecretPermute, $i, ub+11, ub+12, ub+14$)
(OpenSup, $ub+13, ub+19$)
(Destroy, $ub+14, ub+19$)

Q.E.D.

THE REDUNDANCY OF THE DESTROY OPERATION. Notice that the super-envelope $D^{(i,t)}$ generated in Step 5, contains a sequence of 5 envelopes whose contents encode a permutation in SYM_5 —the complement of the tth bit of player i's input. Thus, in Step 7, $D^{(i,t)}$ is destroyed to ensure that no residual information is left on the table. The same assurance can be obtained, without relying on the destroy operation, by replacing Step 7 by the following three steps: (7A) super-envelope $D^{(i,t)}$ is opened so as to reveal the 5 envelopes $\delta_1^{(i,t)}, \ldots, \delta_5^{(i,t)}$; (7B) these 5 envelopes are ballot-boxed to ensure that their contents are no longer correlated with the tth bit of i; and (7C) the randomly permuted 5 envelopes are publicly opened. By doing so, however, the players will observe a public random signal —namely, a random permutation in $\mathbb{S}_5$— which potentially could be used in choosing future input bits in ways that have no counterpart in a normal-form mechanism. To prevent their from happening, for all iterations (i,t), it suffices to execute all Steps 7A–7C after all inputs bits have been committed to.

F.4. Our Basic Ballot-Box Computers

In this section we first provide ballot-box computers for three (auxiliary) elementary functions: namely, Permutation Inverse; Permutation Product; and Permutation Cloning. Then we show how to utilize these auxiliary computers for building computers for $\overline{DUPLICATE}$; $\overline{COIN}$; $\overline{AND}$ and $\overline{NOT}$. As already hinted, these last 4 computers suffice for constructing an arbitrary ballot-box computer. Such final construction is formally presented in Section F.6, after establishing some basic notation for the composition of finite functions.

F.4.1. Our Ballot-Box Computer for Permutation Inverse

Permutation inverse is the function that, on input a permutation $p \in \mathbb{S}_5$, returns p^{-1}.

Protocol $\mathcal{INV}_\sigma$

Input address: σ —an envelope encoding of a permutation in $\mathbb{S}_5$.
(1) Create an envelope encoding α of the identity permutation $\mathcal{I} = 12345$.
(2) For $\ell = 1$ to 5: make a new super-envelope A_ℓ containing the pair of envelopes $(\alpha_\ell, \sigma_\ell)$.
(3) Ballotbox $A_1, \ldots, A_5$ to obtain $A'_1, \ldots, A'_5$.
(4) For $\ell = 1$ to 5: open super-envelope A'_ℓ to expose envelope pair (μ_ℓ, ν_ℓ).
(5) For $\ell = 1$ to 5: publicly open ν_ℓ, and denote its content by $\hat{\nu}_\ell$. Set $\hat{\nu} = \hat{\nu}_1 \circ \cdots \circ \hat{\nu}_5$.
(6) Publicly permute $\mu_1, \ldots, \mu_5$ according to $\hat{\nu}^{-1}$ to obtain $\rho_1, \ldots, \rho_5$. Set $\rho = \rho_1, \ldots, \rho_5$.
Output address: $26, 27, 28, 29, 30$.

LEMMA 2: *For any 5-long address σ, $\mathcal{INV}_\sigma$ is a ballot-box computer for permutation inverse, with input address σ and output address* $26, \ldots, 30$.

Proof. As per Definition 14, let us establish Correctness, Privacy and Clean Operation for $\mathcal{INV}_\sigma$. Consider an execution of $\mathcal{INV}_\sigma$ on any initial memory gm^0 proper for permutation inverse and σ, and let $\mathtt{ub}^0$ be the identifier upper-bound of gm^0.

CORRECTNESS. Step 1 generates 5 new identifiers (increasing $\mathtt{ub}^0$ by 5). Step 2 binds together, in the same super-envelope A_ℓ, the ℓth envelope of α and σ. It generates 5 new identifiers, and all of its actions are feasible since $\sigma \in I_B$. Step 3 applies the same, random and secret, permutation to both $\hat{\alpha}$ and $\hat{\sigma}$, generating 5 new identifiers. Letting x be this secret permutation, Step 4 "puts on the table" the envelope encodings $\mu = \mu_1, \ldots, \mu_5$ and $\nu = \nu_1, \ldots, \nu_5$ where $\hat{\mu} = x\mathcal{I} = x$ and $\hat{\nu} = x\hat{\sigma}$, and generates 10 new identifiers. At the end of Step 4, both $\hat{\nu}$ and $\hat{\mu}$ are totally secret. Step 5, however, reveals $\hat{\nu}$ to all players, so that all players can compute $\hat{\nu}^{-1}$ and verify that Step 6 is correctly executed. The action of Step 6 is feasible because $\hat{\sigma} \in \mathbb{S}_5$ and so $\hat{\nu} \in \mathbb{S}_5$. Step 6 produces five new identifiers: 26th to 30th, counting from the start of the execution. Thus, $\rho = \mathtt{ub}^0 + 26, \ldots, \mathtt{ub}^0 + 30$; and $\hat{\rho} = \hat{\nu}^{-1}\hat{\mu} = \hat{\sigma}^{-1}x^{-1}x = \hat{\sigma}^{-1}$ as desired.

PRIVACY. Consider the following probabilistic algorithm.

Algorithm $SIM\text{–}\mathcal{INV}_\sigma$

Input: ub —an integer.
Randomly select a permutation $r \in \mathbb{S}_5$, and output the following sequence of records (grouped so as to correspond to the records generated by each conceptual step of protocol $\mathcal{INV}_\sigma$):

$(\textsc{NewEn}, 1, ub+1)\ \cdots\ (\textsc{NewEn}, 5, ub+5)$
$(\textsc{NewSup}, ub+1, \sigma_1, ub+6)\ \cdots\ (\textsc{NewSup}, ub+5, \sigma_5, ub+10)$
$(\textsc{BallotBox}, ub+6, \ldots, ub+10, ub+15)$
$(\textsc{OpenSup}, ub+11, ub+17)\ (\textsc{OpenSup}, ub+12, ub+19)\ \cdots\ (\textsc{OpenSup}, ub+15, ub+25)$
$(\textsc{OpenEn}, ub+17, r_1, ub+25)\ (\textsc{OpenEn}, ub+19, r_2, ub+25)\ \cdots\ (\textsc{OpenEn}, ub+25, r_5, ub+25)$
$(\textsc{PublicPermute}, ub+16, ub+18, ub+20, ub+22, ub+24, r^{-1}, ub+30)$

To see that $SIM\text{–}\mathcal{INV}_\sigma$ is the required simulator for $\mathcal{INV}_\sigma$, consider a random execution e of $\mathcal{INV}_\sigma$ on gm^0 and a random output E of $SIM\text{–}\mathcal{INV}_\sigma(\mathtt{ub}^0)$. First observe that the first 16 records of e and E (corresponding to Steps 1-4) are identical. The remaining 6 records of the public record of e are:

$(\textsc{OpenEn}, \mathtt{ub}^0+17, \hat{\nu}_1, \mathtt{ub}^0+25)\ \cdots\ (\textsc{OpenEn}, \mathtt{ub}^0+25, \hat{\nu}_5, \mathtt{ub}^0+25)$
$(\textsc{PublicPermute}, \mathtt{ub}^0+16, \mathtt{ub}^0+18, \mathtt{ub}^0+20, \mathtt{ub}^0+22, \mathtt{ub}^0+24, \hat{\nu}^{-1}, \mathtt{ub}^0+30)$

Thus, while the last 6 records produced by e and E may very well be different, they are identically distributed. In fact, $r = r_1r_2r_3r_4r_5$ is the random permutation selected by the simulator; and $\hat{\nu} = \hat{\nu}_1\hat{\nu}_2\hat{\nu}_3\hat{\nu}_4\hat{\nu}_5 = x\hat{\sigma}$, where x is the random permutation (secretly) selected by the ballot box. Since the product of a random permutation in $\mathbb{S}_5$ and a fixed permutation in $\mathbb{S}_5$ is a random permutation in $\mathbb{S}_5$, public records of e and E are identically distributed.

CLEAN OPERATION. Trivially follows by construction. *Q.E.D.*

F.4.2. Our Ballot-Box Computer for Permutation Product

Permutation product is the function that, on input (p, q), where $p, q \in \mathbb{S}_5$, returns pq —i.e., the permutation in $\mathbb{S}_5$ mapping i to $p(q(i))$.

Protocol $\mathcal{MULT}_{\sigma,\tau}$

Input addresses: σ and τ —each an envelope encoding of a permutation in $\mathbb{S}_5$.

(1) Execute computer $\mathcal{INV}_\sigma$ to obtain the envelope encoding α.

(2) For $\ell = 1$ to 5: make a new super-envelope A_ℓ containing the pair of envelopes (τ_ℓ, α_ℓ).

(3) Ballotbox $A_1, \dots, A_5$ to obtain $A'_1, \dots, A'_5$.

(4) For $\ell = 1$ to 5: open super-envelope A'_ℓ to expose envelope pair (μ_ℓ, ν_ℓ).

(5) For $\ell = 1$ to 5: publicly open envelope ν_ℓ, and denote its content by $\hat{\nu}_\ell$. Set $\hat{\nu} = \hat{\nu}_1 \circ \cdots \circ \hat{\nu}_5$.

(6) Publicly permute μ according to $\hat{\nu}^{-1}$ to obtain ρ.

Output address: $51, 52, 53, 54, 55$.

LEMMA 3: *For any two, disjoint, 5-long addresses σ and τ, $\mathcal{MULT}_{\sigma,\tau}$ is a ballot-box computer for permutation product, with input addresses σ and τ and output address* $51, \dots, 55$.

Proof. Consider an execution of $\mathcal{MULT}_{\sigma,\tau}$ on any initial memory gm^0 proper for permutation product and σ, τ, and let $\mathtt{ub}^0$ be the identifier upper-bound of gm^0.

CORRECTNESS. Note that by Lemma 2, Step 1 generates 30 new identifiers (increasing $\mathtt{ub}^0$ by 30) and produces envelope encoding α with $\hat{\alpha} = \hat{\sigma}^{-1}$. Step 2 binds together, in the same super-envelope A_ℓ, the ℓth envelope of τ and α. It generates 5 new identifiers, and all of its actions are feasible since α is an envelope encoding. Step 3 applies the same, random and secret, permutation to both $\hat{\tau}$ and $\hat{\alpha}$, generating 5 new identifiers. Letting x be this secret permutation, Step 4 "puts on the table" the envelope encodings $\mu = \mu_1, \dots, \mu_5$ and $\nu = \nu_1, \dots, \nu_5$ where $\hat{\mu} = x\hat{\tau}$ and $\hat{\nu} = x\hat{\alpha} = x\hat{\sigma}^{-1}$, and generates 10 new identifiers. At the end of Step 4, both $\hat{\nu}$ and $\hat{\mu}$ are totally secret. Step 5, however, reveals $\hat{\nu}$ to all players, so that all players can compute $\hat{\nu}^{-1}$ and verify that Step 6 is correctly executed. The action of Step 6 is feasible because $\hat{\alpha} \in \mathbb{S}_5$ and so $\hat{\nu} \in \mathbb{S}_5$. Step 6 produces five new identifiers: 51th to 55th, counting from the start of the execution. Thus, $\rho = \mathtt{ub}^0 + 51, \dots, \mathtt{ub}^0 + 55$; and $\hat{\rho} = \hat{\nu}^{-1}\hat{\mu} = \hat{\sigma}x^{-1}x\hat{\tau} = \hat{\sigma}\hat{\tau}$ as desired.

PRIVACY. Consider the following probabilistic algorithm.

Algorithm $SIM\text{-}\mathcal{MULT}_{\sigma,\tau}$

Input: ub —an integer;

Run $SIM\text{–}\mathcal{INV}_\sigma(ub)$; randomly select a permutation $r \in \mathbb{S}_5$; and add to the output the following sequence of records:

$(\text{NEWSUP}, ub + 26, \tau_1, ub + 31) \ \cdots \ (\text{NEWSUP}, ub + 30, \tau_5, ub + 35)$

$(\text{BALLOTBOX}, ub + 31, \dots, ub + 35, ub + 40)$

$(\text{OPENSUP}, ub + 36, ub + 42) \ (\text{OPENSUP}, ub + 37, ub + 44) \ \cdots \ (\text{OPENSUP}, ub + 40, ub + 50)$

$(\text{OPENEN}, ub + 42, r_1, ub + 50) \ (\text{OPENEN}, ub + 44, r_2, ub + 50) \ \cdots \ (\text{OPENEN}, ub + 50, r_5, ub + 50)$

$(\text{PUBLICPERMUTE}, ub + 41, ub + 43, ub + 45, ub + 47, ub + 49, r^{-1}, ub + 55)$

To see that $SIM{-}\mathcal{MULT}_{\sigma,\tau}$ is the required simulator for $\mathcal{MULT}_{\sigma,\tau}$, consider a random execution e of $\mathcal{MULT}_{\sigma,\tau}$ on gm^0 and a random execution E of $SIM{-}\mathcal{MULT}_{\sigma,\tau}(\mathtt{ub}^0)$. Lemma 2 implies that the 22 records produced by protocol $\mathcal{INV}_\sigma$ in conceptual Step 1 of execution e are distributed identically to the first 22 produced by $SIM{-}invert_\sigma(\mathtt{ub}^0)$ in E. Next, 11 records produced by Steps 2-4 of $\mathcal{MULT}_{\sigma,\tau}$ are identical to the next 11 records in E by construction (they are constant).

The remaining 6 records of e and E differ only in the permutation they use, respectively, $\hat{\nu}$ and r. Since both $\hat{\nu}$ and r are random permutations in $\mathbb{S}_5$, these are identically distributed. In addition, random permutations of Step 1 of $SIM{-}\mathcal{INV}_\sigma(\mathtt{ub}^0)$ and of Step 2 of $SIM{-}\mathcal{MULT}_{\sigma,\tau}(\mathtt{ub}^0)$ are selected independently from each other. The same holds for (secret) permutations selected by the ballot-box in Steps 6 of $\mathcal{INV}_\sigma$ and of $\mathcal{MULT}_{\sigma,\tau}$. Therefore, the whole records produced by e and E are identically distributed.

CLEAN OPERATION. Evident from our construction. *Q.E.D.*

F.4.3. Our Ballot-Box Computer for Permutation Clone

Permutation clone is the function that, on input a permutation $p \in \mathbb{S}_5$, returns the pair of permutations (p, p).

Protocol $\mathcal{CLONE}_\sigma$

Input address: σ —an envelope encoding of a permutation in $\mathbb{S}_5$.

(1) Execute computer $\mathcal{INV}_\sigma$ to obtain the envelope encoding α.

(2) Create two envelope encodings, β and γ, of the identity permutation $\mathcal{I}$.

(3) For $\ell = 1$ to 5: make a new super-envelope A_ℓ containing the triple of envelopes $(\beta_\ell, \gamma_\ell, \alpha_\ell)$.

(4) Ballotbox $A_1, \ldots, A_5$ to obtain $A'_1, \ldots, A'_5$.

(5) For $\ell = 1$ to 5: open super-envelope A'_ℓ to expose envelope triple $(\mu_\ell, \nu_\ell, \eta_\ell)$.

(6) For $\ell = 1$ to 5: publicly open envelope η_ℓ, and denote its content by $\hat{\eta}_\ell$. Set $\hat{\eta} = \hat{\eta}_1 \circ \cdots \circ \hat{\eta}_5$.

(7) Publicly permute μ and ν according to $\hat{\eta}^{-1}$.[31]

Output addresses: $66, 67, 68, 69, 70$ and $71, 72, 73, 74, 75$.

LEMMA 4: *For any 5-long address σ, $\mathcal{CLONE}_\sigma$ is a ballot-box computer for permutation clone, with input address σ and output addresses* $66, \ldots, 70$ *and* $71, \ldots, 75$.

Proof. In light of the previous two proofs, this one is quite straightforward. The simulator required by Privacy is the probabilistic algorithm below.

Algorithm $SIM{-}\mathcal{CLONE}_\sigma$

Input: ub —an integer.

(1) Run $SIM{-}\mathcal{INV}_\sigma(ub)$;

(2) Randomly select a permutation $r \in \mathbb{S}_5$ and add to the output the following sequence of records:

$$(\textsc{NewEn}, 1, ub + 31) \cdots (\textsc{NewEn}, 5, ub + 35) \qquad (\textsc{NewEn}, 1, ub + 36) \cdots (\textsc{NewEn}, 5, ub + 40)$$

[31]Note that Steps 2–7, in essence, correspond to a protocol for permutation inverse that on input α produces two identical envelope encodings, each encoding $\hat{\alpha}^{-1}$.

$(\textsc{NewSup}, ub+26, ub+31, ub+36, ub+41)$ $\cdots$ $(\textsc{NewSup}, ub+30, ub+35, ub+40, ub+45)$
$(\textsc{BallotBox}, ub+41, \ldots, ub+45, ub+50)$
$(\textsc{OpenSup}, ub+46, ub+53)$ $(\textsc{OpenSup}, ub+47, ub+56)$ $\cdots$ $(\textsc{OpenSup}, ub+50, ub+65)$
$(\textsc{OpenEn}, ub+53, r_1, ub+65)$ $(\textsc{OpenEn}, ub+56, r_2, ub+65)$ $\cdots$
$(\textsc{OpenEn}, ub+65, r_5, ub+65)$
$(\textsc{PublicPermute}, ub+51, ub+54, ub+57, ub+60, ub+63, r^{-1}, ub+70)$
$(\textsc{PublicPermute}, ub+52, ub+55, ub+58, ub+61, ub+64, r^{-1}, ub+75)$

Q.E.D.

Corollary 1: *For any 5-long address σ, $\mathcal{CLONE}_\sigma$ is a ballot-box computer for $\overline{DUPLICATE}$ with input address σ, output addresses $66, \ldots, 70$ and $71, \ldots, 75$, and simulator $SIM{-}\mathcal{CLONE}_\sigma$.*

Proof. Recall that $DUPLICATE$ is the binary function that, on input a bit b, outputs the pair of bits (b, b). Thus, a ballot-box computer for $\overline{DUPLICATE}$ must, on input an envelope encoding of a bit b, return two envelope encodings of the same bit. This is exactly what $\mathcal{CLONE}_\sigma$ does. In fact, (1) when σ is an envelope encoding of the identity permutation of $\mathbb{S}_5$, then so are ρ and ψ; and (2) when σ is an envelope encoding of $a = 12453$, then so are ρ and ψ.

Q.E.D.

F.4.4. Our Ballot-Box Computer for $\overline{COIN}$

Recall that $COIN$ is the probabilistic function that, on no input, returns a random bit. Accordingly, $\overline{COIN}$ is the probabilistic function that, on empty input, returns either $\mathcal{I}$ or a with equal probability.

Protocol $\mathcal{COIN}$

(1) Create an envelope encoding α of $\mathcal{I}$ and an envelope encoding β of a.
(2) Make new super-envelopes A and B containing envelopes $\alpha_1, \ldots, \alpha_5$ and $\beta_1, \ldots, \beta_5$, respectively.
(3) Ballotbox A and B to obtain super-envelopes C and D.
(4) Open C to expose an envelope encoding γ. Destroy D.

Output address: $15, 16, 17, 18, 19$.

Lemma 5: *Protocol $\mathcal{COIN}$ is a ballot-box computer $\overline{COIN}$, with no input address and output address $15, \ldots, 19$.*

Proof. The only non-trivial part to prove Correctness is to demonstrate that contents of γ are random and belong to $\{\mathcal{I}, a\}$. Indeed, at the end of Step 2, A contains a sequence of 5 envelopes encoding $\mathcal{I}$, and B contains a sequence of 5 envelopes encoding permutation a. At the end of Step 3, the contents of C are either those of A or of B with equal probabilities. Thus, at the end of Step 4, the content of address γ is random and is either $\mathcal{I}$ or a.

Clean Operation is trivial, and Privacy straightforwardly follows by noting that the required simulator for $\mathcal{COIN}$ is the following probabilistic algorithm.

Algorithm $SIM{-}\mathcal{COIN}$

Input: ub —an integer.
Output the following sequence of records:

$(\textsc{NewEn}, 1, ub+1)$ $(\textsc{NewEn}, 2, ub+2)$ $(\textsc{NewEn}, 3, ub+3)$ $(\textsc{NewEn}, 4, ub+4)$ $(\textsc{NewEn}, 5, ub+5)$ $(\textsc{NewEn}, 1, ub+6)$ $(\textsc{NewEn}, 2, ub+7)$ $(\textsc{NewEn}, 4, ub+8)$ $(\textsc{NewEn}, 5, ub+9)$ $(\textsc{NewEn}, 3, ub+10)$ $(\textsc{NewSup}, 1, 2, 3, 4, 5, ub+11)$ $(\textsc{NewSup}, 6, 7, 8, 9, 10, ub+12)$
$(\textsc{BallotBox}, 11, 12, ub+14)$ $(\textsc{OpenSup}, 13, ub+14)$ $(\textsc{Destroy}, 14, ub+14)$

Q.E.D.

F.4.5. Our Ballot-Box Computer for $\overline{NOT}$

Protocol $\mathcal{NOT}_\sigma$

Input address: σ —$\mathbb{S}_5$ encoding of a bit.

(1) Create an envelope encoding α of permutation 12543.
(2) Execute computer $\mathcal{MULT}_{\sigma,\alpha}$ to obtain the envelope encoding β.
(3) Publicly permute β according to 12354.

Output address: $61, 62, 63, 64, 65$.

LEMMA 6: *For any 5-long address σ, $\mathcal{NOT}_\sigma$ is a ballot-box computer for $\overline{NOT}$, with input address σ and output address* $61, \ldots, 65$.

Proof. Correctness trivially follows from the realization of the Boolean function NOT of Section F.1.1: namely, if σ is the $\mathbb{S}_5$ encoding of a bit b, then $12354\,\sigma\,12543$ encodes $\neg b$. Clean Operation is self evident, and Privacy is trivially established by noting that the required simulator is the following probabilistic algorithm.

Algorithm $SIM\text{–}\mathcal{NOT}_\sigma$

Input: ub —an integer.

(1) Output $(\textsc{NewEn}, 1, ub{+}1)$ $(\textsc{NewEn}, 2, ub{+}2)$ $(\textsc{NewEn}, 5, ub{+}3)$ $(\textsc{NewEn}, 4, ub{+}4)$ $(\textsc{NewEn}, 3, ub{+}5)$.
    Set $\alpha = ub+1, \ldots, ub+5$.
(2) Run $SIM\text{–}\mathcal{MULT}_{\sigma,\alpha}(ub+5)$. Set $\beta = ub+56, \ldots, ub+60$.
(3) Add to the current output the record $(\textsc{PublicPermute}, \beta, 12354, ub+65)$.

Q.E.D.

F.4.6. Our Ballot-Box Computer for $\overline{AND}$

Protocol $\mathcal{AND}_{\sigma,\tau}$

Input addresses: σ, τ —each $\mathbb{S}_5$ encoding of a bit.

(1) Create an envelope encoding α of permutation 24351.
(2) Execute computer $\mathcal{CLONE}_\sigma$ to obtain the pair of envelope encodings (σ', σ'').
    Execute computer $\mathcal{CLONE}_\tau$ to obtain the pair of envelope encodings (τ', τ'').

(3) Execute computer $\mathcal{INV}_{\tau''}$ to obtain the envelope encoding β.
Execute computer $\mathcal{INV}_{\sigma''}$ to obtain the envelope encoding γ.

(4) Execute computer $\mathcal{MULT}_{\beta,\alpha}$ to obtain the envelope encoding δ.
Publicly permute δ according to 34125 to obtain κ.

(5) Execute computer $\mathcal{MULT}_{\gamma,\kappa}$ to obtain the envelope encoding μ.
Publicly permute μ according to 34125 to obtain π.

(6) Execute computer $\mathcal{MULT}_{\tau',\pi}$ to obtain the envelope encoding λ.
Publicly permute λ according to 34125 to obtain ξ.

(7) Execute computer $\mathcal{MULT}_{\sigma',\xi}$ to obtain the envelope encoding ν.
Publicly permute ν according to 32514.

Output address: $451, 452, 453, 454, 455$.

<u>LEMMA</u> 7: *For any two, disjoint, 5-long addresses σ and τ, $\mathcal{AND}_{\sigma,\tau}$ is a ballot-box computer for $\overline{AND}$, with input addresses σ and τ and output address* $451, \ldots, 455$.

Proof. Correctness trivially follows from the realization of the Boolean function AND of Section F.1.1: namely, if σ and τ respectively are $\mathbb{S}_5$ encodings of bits c and d, then the $\mathbb{S}_5$ encoding of $c \wedge d$ is $32514\ \sigma\ 34125\ \tau\ 34125\ \sigma^{-1}\ 34125\ \tau^{-1}\ 24351$. Clean Operation is self evident and Privacy follows from realizing that the required simulator is the following probabilistic algorithm.

Algorithm $SIM\!-\!\mathcal{AND}_{\sigma,\tau}$

Input: ub —an integer.

(1) Output (NEWEN, $1, ub+1$) (NEWEN, $2, ub+2$) (NEWEN, $5, ub+3$) (NEWEN, $4, ub+4$) (NEWEN, $3, ub+5$). Set $\alpha = ub+1, \ldots, ub+5$.

(2) Run $SIM\!-\!\mathcal{CLONE}_\tau(ub+5)$ and $SIM\!-\!\mathcal{CLONE}_\sigma(ub+80)$.[32]
Set $\sigma' = ub+71, \ldots, ub+75$, $\sigma'' = ub+76, \ldots, ub+80$, $\tau' = ub+146, \ldots, ub+150$, and $\tau'' = ub+151, \ldots, ub+155$.

(3) Run $SIM\!-\!\mathcal{INV}_{\tau''}(ub+155)$ and $SIM\!-\!\mathcal{INV}_{\sigma''}(ub+185)$.
Set $\beta = ub+181, \ldots, ub+185$ and $\gamma = ub+211, \ldots, ub+215$.

(4) Run $SIM\!-\!\mathcal{MULT}_{\beta,\alpha}(ub+215)$. Set $\delta = ub+266, \ldots, ub+270$. Add to the current output the record (PUBLICPERMUTE, $\delta, 34125, ub+275$); and set $\kappa = ub+271, \ldots, ub+275$.

(5) Run $SIM\!-\!\mathcal{MULT}_{\gamma,\kappa}(ub+275)$. Set $\mu = ub+326, \ldots, ub+330$. Add to the current output the record (PUBLICPERMUTE, $\mu, 34125, ub+335$). Set $\pi = ub+331, \ldots, ub+335$.

(6) Run $SIM\!-\!\mathcal{MULT}_{\tau',\pi}(ub+335)$. Set $\lambda = ub+386, \ldots, ub+390$. Add to the current output the record (PUBLICPERMUTE, $\lambda, 34125, ub+395$). Set $\xi = ub+391, \ldots, ub+395$.

(7) Run $SIM\!-\!\mathcal{MULT}_{\sigma,\xi}(ub+395)$. Set $\nu = ub+446, \ldots, ub+450$. Add to the current output the record (PUBLICPERMUTE, $\nu, 32514, ub+455$).

Q.E.D.

Therefore, protocols $\mathcal{NOT}$ and $\mathcal{AND}$ for functions $\overline{NOT}$ and $\overline{AND}$ can be easily constructed by creating envelope encodings of constant permutations, and, in the correct order, executing protocols $\mathcal{INV}$, $\mathcal{MULT}$, $\mathcal{CLONE}$, and publicly permuting a set of 5 envelopes for each permutation product where the left permutation is a known constant.

[32]Recall that protocols $\mathcal{INV}$, $\mathcal{MULT}$, and $\mathcal{CLONE}$ generate, respectively, 30, 55, and 75 new identifiers.

REMARK. Notice that $\mathcal{AND}$ and $\mathcal{NOT}$ take as inputs envelope encodings of bits and produce envelope encodings of bits. By contrast, the ballot-box protocols $\mathcal{INV}$, $\mathcal{MULT}$, and $\mathcal{CLONE}$ (that underly our construction of $\mathcal{NOT}$ and $\mathcal{AND}$) are ballot-box computers operating on envelope encodings of arbitrary permutations in $\mathbb{S}_5$, rather than just 12345 or 12453.

F.5. Our Ballot-Box Revealers

If the outcome function g of a normal-form mechanism $\mathcal{M}$ produces L-bit outputs, then the $\mathbb{S}_5$ encoding $\bar{g}$ produces outputs of length $5L$. Therefore, the revealer of a ballot-box implementation of $\mathcal{M}$ must open $5L$ envelopes.

Protocol $Reveal_y$

Input address: $y = y_1, \ldots, y_L$ —where each y_i is a sequence of 5 envelope identifiers.

```
For $i = 1$ to $L$ do: for $j = 1$ to 5 do: publicly open envelope $y_i^j$.
```

F.6. Our Computers for General Finite Functions

To implement any given normal form mechanism, it is much easier to construct its ballot-box committer and revealer than its ballot-box computer. Indeed, committers and revealers have a fixed functionality, and thus "designing one is designing them all." By contrast, there are "infinitely diverse" outcome functions, and in principle designing a ballot-box computer for one of them may provide negligible help in designing a ballot-box computer for another. Fortunately, standard complexity theoretic results state that any finite function g is equivalent to a *Boolean circuit* —in essence, a "special composition" of the 4 Boolean functions NOT, AND, $DUPLICATE$, and $COIN$.[33] We exploit this fact to construct a ballot-box computer for any given outcome function from the already constructed ballot-box computers for (the $\mathbb{S}_5$ encodings of) these elementary functions.

F.6.1. Boolean Circuits

DIRECTED GRAPHS. A directed graph —*digraph* for short— consists of a pair (V, E), where V is a finite set and E a subset of $V \times V$. We refer to V as the set of *nodes*, and to E as the set of *edges*. Whenever $(u, v) \in E$, we refer to u as a *parent* of v, and to v as a *child* of u. We refer to the number of parents of a node u as u's *in-degree*, and to the number of children of u as u's *out-degree*. A node of V is called a *source* if it has in-degree zero, a *sink* if it has out-degree zero, and an *internal node* otherwise.

A *path* is a sequence of nodes $s_1, \ldots, s_k$ such that for all $i < k$, $(s_i, s_{i+1}) \in E$. A digraph (V, E) is *acyclic* if, for any path $s_1, \ldots, s_k$ such that $k > 1$, $s_1 \neq s_k$. (Note that any non-empty acyclic digraph has at least one source and at least one sink.) In an acyclic digraph, the *maxheight* of a node u, denoted by $maxheight(u)$, is the maximum integer k for which there exists a path $s_1, \ldots, s_k$ in which s_1 is a source and $s_k = u$.

For an acyclic digraph with n nodes, a function $ord : V \to \{1, \ldots, n\}$ is a *natural order* if

[33] At the simplest level, a function h is the composition of two functions f and g if $h(x) = f(g(x))$ for all x. Unfortunately this simple type of composition does not guarantee that any function is the composition of a few simple ones, and we thus need to work with more general compositions.

1. for any source u, internal node v, and sink z, $ord(u) < ord(v) < ord(z)$; and
2. for any internal nodes u and v, $ord(u) < ord(v)$ whenever $maxheight(u) < maxheight(v)$.

When a natural order ord is specified, by *"node j"* we mean the node u such that $ord(u) = j$.

BOOLEAN CIRCUITS. A *Boolean Circuit* C consists of a quadruple $((V, E), ord, a, Func)$ where:

- (V, E) is an acyclic digraph such that every sink has maxheight ≥ 3 and
  - every source has out-degree one and every sink has in-degree one;
  - every internal node has in-degree 1 or 2 and out-degree 1 or 2; but
  - no node has in-degree 2 and out-degree 2.
- ord is a natural order for V;
- a is a positive integer not exceeding the number of sources in (V, E); and
- $Func$ is a function mapping the internal nodes and the sources whose order is greater than a to the set of functions $\{NOT, AND, DUPLICATE, COIN\}$ as follows:
  - if node j is a source, then $Func(j) = COIN$;
  - if node j has in-degree 1 and out-degree 1, then $Func(j) = NOT$;
  - if node j has in-degree 2 and out-degree 1, then $Func(j) = AND$; and
  - if node j has in-degree 1 and out-degree 2, then $Func(j) = DUPLICATE$.

We refer to the first a sources as *input nodes*, to all other sources and internal nodes as *computation nodes,* and to the sinks as *output nodes.*

GRAPHICAL REPRESENTATION OF BOOLEAN CIRCUITS. A Boolean circuit $((V, E), ord, a, Func)$ can be represented graphically as follows: (1) each node u is represented by a "separate" circle, shrinking the circles of input and output nodes to a "dot"; (2) ord is represented by drawing the circle of node u "lower than or on the left of" the circle of node v whenever $ord(u) < ord(v)$; (3) each edge (u, v) is as an arrow directed from the circle representing u to the circle representing v; and (4) for each computation node j, $Func(j)$ appears inside the circle representing node j. For brevity, we denote NOT by $\neg$, AND by $\wedge$, $DUPLICATE$ by $\mathbb{D}$, and $COIN$ by \$. See Figure 1.

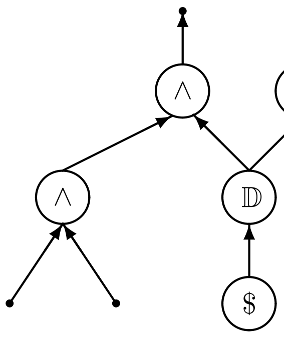

Figure 1: Graphical representation of a Boolean circuit with two input nodes.

F.6.2. Boolean Circuits as Functions

Let $C = ((V, E), ord, a, Func)$ be a Boolean circuit with a input nodes, b output nodes, and c computation nodes. Then C can be interpreted as the finite function $\mathcal{F}_C : \{0,1\}^a \to \{0,1\}^b$

mapping an input vector $x = x_1 \dots x_a$ to an output vector $y = y_1 \dots y_b$ as follows:

- For $i = 1$ to a, label the outgoing edge of the ith source with bit x_i.
- For $i = a + 1$ to $a + c$ do:
  if $Func(i) = COIN$, then randomly choose a bit b and label i's out-going edge by b;
  if $Func(i) = NOT$ and bit b labels i's incoming edge, label i's outgoing edge by $\neg b$;[34]
  if $Func(i) = DUPLICATE$ and b labels i's incoming edge, label i's outgoing edges by b;
  if $Func(i) = AND$ and b_1, b_2 label i's incoming edges, label i's outgoing edge by $b_1 \wedge b_2$.
- For $i = 1$ to b, set y_i to be the bit labeling the incoming edge of (sink) node $a + c + i$.

We say that function $\mathcal{F}_C$ is *deterministic* if $Func$ does not map any computation node to $COIN$; and *probabilistic* otherwise.

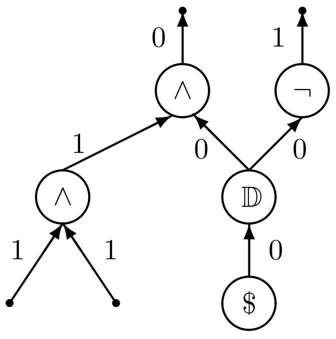

Figure 2: A computation of the Boolean circuit of Figure 1, on input 11 and coin toss 0.

BOOLEAN-CIRCUIT REPRESENTATION OF FINITE FUNCTIONS. A Boolean circuit representation of a finite function f is a Boolean circuit C such that $f = \mathcal{F}_C$. It is well known in complexity theory that a Boolean-circuit representation of f can be efficiently computed from any other standard representation of f.

F.6.3. Constructing Ballot-Box Computers for Boolean Circuits.

Intuitively, we construct a ballot-box computer for any Boolean circuit $C = ((V, E), ord, a, Func)$ by simply "translating" C into a protocol $\mathcal{P}_C$. As we have seen above, the function $\mathcal{F}_C$ "visits each computation node j according to ord, evaluates $Func(j)$ on the bits found on j's incoming edges, and passes the outputs along j's outgoing edges of (V, E)." Similarly, $\mathcal{P}_C$ "visits each computation node j in the same order, and executes the corresponding ballot-box computer for $\overline{Func(j)}$ on the input envelopes found on j's incoming edges, and passes the output envelopes along j's outgoing edges." Technically, $\mathcal{P}_C$ is a concatenation of (multiple copies of) $\mathcal{NOT}$, $\mathcal{AND}$, $\mathcal{CLONE}$, and $\mathcal{COIN}$, in proper order and with proper input addresses.

The following algorithm $\mathbb{P}$ formalizes how circuit C is translated into the desired ballot-box computer $\mathcal{P}_C$. Algorithm $\mathbb{P}$ uses $\texttt{ub}_0$ to refer to the identifier upper-bound of the initial global memory $gm^0 = (B^0, R^0, H^0)$ for $\mathcal{P}_C$. (Recall that, when executed on gm^0, the action functions of $\mathcal{P}_C$ can deduce $\texttt{ub}^0$ from R^0.) Algorithm $\mathbb{P}$ constructs $\mathcal{P}$ as the concatenation of ballot-box computers, one

[34]Note: because ord is a natural order for V, when it comes time to label node i's outgoing edges, all of node i's incoming edges have already been labeled.

for each computation node of C. If j is such a computational node, the corresponding ballot-box protocol is denoted by $\mathcal{P}_j$, and the variable w_j predicts the value of the identifier upper-bound at the start of the execution of $\mathcal{P}_j$.

Algorithm $\mathbb{P}$

```
Inputs:
    $C = ((V, E), ord, a, Func)$ —a Boolean Circuit with $b$ output nodes and $c$ computation nodes;
    $x = x_1, \ldots, x_a$ —an address such that each $x_i$ has length 5.
 (1) Set $w_{a+1} = \mathtt{ub}^0$.
 (2) For $i = 1$ to $a$, label the outgoing edge of the $i$th source with $x_i$.
 (3) For $i = a+1$ to $a+c$, do:
     if $Func(i) = COIN$,
     then set $\mathcal{P}_i = \mathcal{COIN}$; label $i$'s outgoing edge with $\sigma_i = 15 + w_i, \ldots, 19 + w_i$; and set $w_{i+1} =$
         $w_i + 19$;
     if $Func(i) = NOT$ and $\sigma$ labels $i$'s incoming edge
     then set $\mathcal{P}_i = \mathcal{NOT}_\sigma$; label $i$'s outgoing edge with $\sigma_i = 61 + w_i, \ldots, 65 + w_i$; and set $w_{i+1} =$
         $w_i + 65$;
     if $Func(i) = AND$ and $(\sigma; \tau)$ label $i$'s incoming edges,
     then set $\mathcal{P}_i = \mathcal{AND}_{\sigma,\tau}$; label $i$'s outgoing edge with $\sigma_i = 451 + w_i, \ldots, 455 + w_i$; set $w_{i+1} =$
         $w_i + 455$;
     if $Func(i) = DUPLICATE$ and $\sigma$ labels $i$'s incoming edge,
     then set $\mathcal{P}_i = \mathcal{CLONE}_\sigma$; label $i$'s outgoing edges with $\sigma_i = 61 + w_i, \ldots, 65 + w_i$ and $\tau_i =$
         $66 + w_i, \ldots, 70 + w_i$; and set $w_{i+1} = w_i + 70$;
 (4) For $i = 1$ to $b$: if $\alpha$ labels the incoming edge of sink $a + c + i$, then set $y_i = \alpha - w^0$;
 (5) Set $\mathcal{P}_C = \mathcal{P}_{a+1} \circ \cdots \circ \mathcal{P}_{a+c}$.
Outputs:
    $\mathcal{P}_C$ —a ballot-box protocol (the desired protocol); and
    $y = y_1, \ldots, y_b$ —an address ($\mathcal{P}_C$'s output address).
```

Lemma 8: *If $\mathbb{P}(C, x) = (\mathcal{P}_C, y)$, where $C = ((V, E), ord, a, Func)$ is a Boolean circuit and $x = \underline{x_1}, \ldots, x_a$ an address such that each x_i has length 5, then protocol $\mathcal{P}_C$ is a ballot-box computer for $\mathcal{F}_C$ with input address x and output address y.*

Proof. The correctness of $\mathcal{P}_c$ follows from the following facts. First, observe that if $\mathtt{ub}^0$ is the identifier upper bound of the initial memory gm^0, proper for $\overline{\mathcal{F}_C}$ and $x = x_1 \ldots x_a$, then for each computation node j, w_j is the initial identifier upper-bound at the start of execution of $\mathcal{P}_j$. Indeed, $w_{a+1} = \mathtt{ub}^0$, and w_{j+1} is generated off-setting w_j by exactly the number of new identifiers created by any execution of its corresponding computer $\mathcal{P}_j$. (Recall that $\mathcal{NOT}$ always increases $\mathtt{ub}$ by 65, $\mathcal{AND}$ by 455, $\mathcal{CLONE}$ by 70, and $\mathcal{COIN}$ by 19.) Second, $\mathbb{P}$ labels each edge with 5 distinct integers. Third, due to the correctness of $\mathcal{NOT}$, $\mathcal{AND}$, $\mathcal{CLONE}$, and $\mathcal{COIN}$ and the order in which $\mathcal{P}_C$ calls for their execution: (1) at the start of $\mathcal{P}_j$'s execution, the label already assigned by $\mathbb{P}$ to each incoming edge of computation node j is guaranteed to be an envelope encoding of a bit; (2) upon termination of $\mathcal{P}_j$, the label already assigned by $\mathbb{P}$ to each outgoing edge of j also is an envelope encoding of a bit; and finally (3) the "underlying" output bit is that (bits are those) of

$Func(j)$ evaluated on the "underlying" input bit(s). Thus, an execution of $\mathcal{P}_C$ labels each edge in E with an envelope encoding of a bit so that the underlying bits of these encodings label (V, E) exactly as does a computation of $\mathcal{F}_C$.

Privacy is easily established by noting that the following probabilistic algorithm $SIM\text{-}\mathcal{P}_C$ is the required simulator for $\mathcal{P}_C$.

$SIM\text{-}\mathcal{P}_C$

Input: ub —an integer.
Set $ub^0 = ub$ and, for each $i = a+1$ to $a+c$, define ub_i as in $\mathbb{P}$.
(1) For $i = a+1$ to $a+c$, run $SIM\text{-}\mathcal{P}_i(ub_i)$.
Clean Operation trivially holds for $\mathcal{P}_C$ as it holds for each individual ballot-box computer $\mathcal{P}_j$.

Q.E.D.

F.7. *Constructing Perfect Implementations For All Normal-Form Mechanisms*

We are finally ready to state and prove our main theorem.

THEOREM 1: *For any standard normal-form mechanism, there exists a ballot-box mechanism $\mathcal{B}$ that perfectly implements it.*

Proof. Let $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$ be a standard normal-form mechanism. In light of Definition 16, to present a ballot-box mechanism $\mathcal{B} = (N, K, J, PS, AF, OF)$ that perfectly implements $\mathcal{M}$, we must specify a bit-by-bit encoding $\{\bar{0}, \bar{1}\}$, and the three ballot-box protocols $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$. We do so as follows:

- Set $\{\bar{0}, \bar{1}\} = \{12345, 12453\}$.
- Set $\mathcal{P}_1 = Commit_L$; and let x and $\mathtt{ub}^*$ respectively be $\mathcal{P}_1$'s output address and final identifier upper bound.
- Set $\mathcal{P}_2 = \mathcal{P}_C$, where C is a Boolean circuit for g and $(\mathcal{P}_C, y) = \mathbb{P}(C, x)$.
- Set $\mathcal{P}_3 = Reveal_{y+\mathtt{ub}^*}$.

Q.E.D.

F.8. *Universality and Efficiency of Our Construction*

Let us now argue that the construction of Section F.7 can be actually carried out in an automatic and efficient way —indeed in linear time— and produces ballot-box mechanisms that themselves work in linear time.

F.8.1. *Mechanism Representation and Definition of Efficiency*

To discuss the efficiency of our construction we must first explain

1. What it means for an algorithm to be linear-time;
2. How a normal-form mechanism is represented as an input to an algorithm;
3. How a ballot-box mechanism $\mathcal{B}$ is represented as an output of an algorithm; and
4. What it means for a ballot-box mechanism to be linear-time.

EXPLANATION 1. The traditional model for analyzing the precise efficiency of concrete algorithms is the *Random-Access Memory (RAM) machine.*[35] In essence, this is the idealization of a modern computer, where memory cells are large, plentiful and easily accessible.[36] In our context, a RAM machine guarantees that each element of a public record (and thus each identifier) can be stored in a single cell of memory and that, after writing the address of a memory cell, the contents of the specified cell can be retrieved in unit time. A RAM machine also guarantees that any elementary operation on the values stored in a constant number of memory cells has a constant computation cost. (Examples of such elementary operations are comparing two values, multiplying a value by a constant, or adding two values.) A different source of computational cost is that of generating an output. Since this is not an internal operation on the "fast and capacious" memory cells of the RAM machine, outputting an k-bit string is defined to take k computational steps.

With this preamble, saying that an algorithm $\mathbb{A}$ runs in linear time means that there exists a positive constant c such that, whenever $\mathbb{A}$'s input is a binary string of length λ,[37] $\mathbb{A}$ terminates within $c\lambda$ steps of computation on a RAM machine.

EXPLANATION 2. Note that a standard normal-form mechanism $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$ can be fully described by a Boolean circuit $C = ((V,E), ord, a, Func)$ computing g.[38] (In fact, all elements of $\mathcal{M}$ are derivable from this representation, because C has nL sources and L sinks.) Let us now provide a specific way to encode C (and thus $\mathcal{M}$). Let C have v nodes and e edges. Then, v and a are represented in unary (i.e., by the concatenation of, respectively, v and a 1s, denoted by 1^v and 1^a). Node j (i.e., the jth node according to ord) is described by a unique l-bit string, where $l = \lceil \log(v) \rceil$: namely, the l-bit binary expansion of integer j (with leading 0s if necessary). Each edge (i,j) of E is described by a unique $2l$-bit string: namely, the concatenation of the descriptions of i and j. We thus define the *standard binary encoding of* C to be the string 1^v01^a0S, where string $S \in \{0,1\}^{2le}$ is the concatenation of the descriptions (in lexicographic order) of the e edges of E. Notice that such encoding can be parsed in a unique way, and that, denoting its length by λ, we have $\lambda = O(v \log v)$.[39] We call such an encoding of C an *E2-representation of* $\mathcal{M}$.

EXPLANATION 3. A ballot-box mechanism $\mathcal{B} = (N, J, K, PS, AF, OF)$ is represented as a bit string specifying the cardinality n of N and integers J and K in binary; the sequence PS as a $K\lceil \log(n) \rceil$-bit string; the sequence $AF = AF^1, \ldots, AF^K$ as K RAM-machine programs; and the function OF as a RAM machine program. We call such a representation of $\mathcal{B}$, an *E3-representation of* $\mathcal{B}$.

An $E3$-representation allows some latitude in choosing the RAM algorithms for AF and OF (a choice that should be made wisely so as to ensure the desired "linearity" of $\mathcal{B}$).

[35]The general notion of an "efficient algorithm" (technically, that of a *polynomial-time algorithm*) is quite independent of the underlying computational model. However, more precise efficiency notions, such as "linear-time" are meaningful only relative to a specific model of computation.

[36]See Cormen, Leiserson, Rivest, and Stein (2001) for a convincing explanation and justification of this model.

[37]Representing inputs in binary prevents wasteful algorithms from being deemed efficient. For example, iterated addition is not an efficient way to multiply integers, but it would be if the input integers were represented in unary — i.e., if integer x were represented by a string of x 1s. Any alphabet with more than one symbol allows for exponentially more compact representation of inputs, which is a big jump with respect to unary representation. But adopting any alphabet with more than two symbols only shortens the input length a mere constant factor relative to the binary alphabet.

[38]Recall that C is easily computable from any other standard representation of g, although not necessarily in linear time.

[39]Indeed, each node of (V,E) has at most 3 edges coming into or out of it. Therefore the number of edges, e, is upper-bounded by $1.5v$. Consequently, the length of the encoding string, $v + 1 + a + 1 + 2le$, is at most $5v\lceil \log(v) \rceil$.

EXPLANATION 4. The running time of a ballot-box mechanism $\mathcal{B}$ does not reflect the effort required by the players to choose their strategies in $\mathcal{B}$.[40] Rather, we define it to be the maximum number of steps required by a player, taken over all possible strategy profiles, and over all possible executions of such strategies. Now, there are two kinds of steps for a player i in an execution of $\mathcal{B}$:

1. *Computational steps.* At any round k in which he is the acting player, the computational steps of i consist of two types:
   1.1 the steps necessary to run program AF^k on the public record R^k, so as to compute the action set A_k available to him, and
   1.2 the steps necessary to run his strategy to choose an action in A_k.
2. *Physical steps.* After i has selected an action in his action set, the physical steps of i are those necessary for him to carry out the chosen ballot-box action on the current set of ballots.

The computational steps are assumed to be carried out in the RAM model, and counted accordingly. As for the physical steps, we count a ballot-box action involving k ballots as k steps.

With this clarification, we say that $\mathcal{B}$ is *linear-time* if there exists a positive constant c such that, letting Λ be the number of bits in an E_3-representation of $\mathcal{B}$, in any execution of $\mathcal{B}$ the total number of steps required from each player is upper-bounded by $c\Lambda$.

Note that, if $\mathcal{B}$ is a perfect implementation of a normal-form mechanism $\mathcal{M}$, then $\mathcal{B}$ is linear-time if and only if, in each execution, each player performs $O(\Lambda)$ computational steps of Type 1.1. Indeed, since each action set of i in $\mathcal{B}$ consists of either (a) a single action or (b) two complementary actions, a strategy of i coincides with binary string m_i and requires trivial computation. (That is, the j's time the action set consists of two complementary actions, i's strategy prescribes the 0-action if the jth bit of m_i is 0, and the 1-action otherwise.) As for physical actions, at most Λ ballot-box actions are performed in any execution of a Λ-bit $\mathcal{B}$. Since any such action manipulates at most 10 distinct ballots, the number of physical steps of $\mathcal{B}$ is guaranteed to be linear in Λ.

F.8.2. Our Construction as an Efficient Compiler

Let us now prove that our construction can be viewed as an efficient compiler $\mathbb{C}$ translating normal-form mechanisms to equivalent and efficient ballot-box ones.

REMARK. Note that, for any L, σ, τ, and y, each action function AF^k of any of the protocols $Commit_L$, $Reveal_y$, $\mathcal{NOT}_\sigma$, $\mathcal{AND}_{\sigma,\tau}$, $\mathcal{CLONE}_\sigma$, and $\mathcal{COIN}$ can be specified as a constant-time RAM program that, on input the last 5 entries of the current public record R, generates the action or the pair of complementary actions of round k. This is quite obvious for $Commit_L$ and $Reveal_y$, and holds also for our 4 basic computers because, for each one of them,

- the number of rounds is fixed;
- the action function of each individual round specifies an action set consisting of a single action;
- for any given round k, the specified action is always from the same class; and
- the arguments of the specified action for each k depend only on (1) address x, (2) round k, (3) the last entries of the public record —the maximum number of such entries is 5 for the case of the PUBLICPERMUTE action— and (4) the initial identifier upper-bound $\mathtt{ub}^0$ —itself trivially computable from k and the current $\mathtt{ub}$ appearing in the last entry of the public record.

[40]The time the players may invest in selecting their strategies in $\mathcal{B}$ essentially coincides with that they may invest in $\mathcal{M}$, if $\mathcal{B}$ perfectly simulates $\mathcal{M}$, and we regard it as "orthogonal" to the efficiency of $\mathcal{B}$ proper.

Theorem 3: *There exists a linear-time algorithm $\mathbb{C}$ that, on input any standard normal-form mechanism $\mathcal{M}$, outputs a linear-time ballot-box mechanism $\mathcal{B}$ that perfectly implements $\mathcal{M}$.*

Proof. Let C be the standard binary representation of the outcome function g of a normal-form mechanism $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$; and let λ be the length of C.

Let us start by proving that $Commit_L$ and its output address x can be constructed in time linear in λ. This is so because

- L and nL, respectively the number of sinks and input nodes, can be computed from the standard binary encoding of C in time linear in λ.
- Protocol $Commit_L$ consists of $15nL$ rounds: indeed it is the repetition nL times of a core of 15 actions (organized in 7 conceptual steps in Section F.3). Since $Commit_L$ always starts with an empty initial memory, its first 15 actions are constant, and thus AF^1 through AF^{15} are constant-time RAM programs themselves computable in constant time. These 15 actions always generate 19 new ballot identifiers. The next 15 actions can be computed from AF^1 to AF^{15} by simply offsetting by 19 all of their identifiers. Thus they too can be computed in constant time on a RAM machine. By continuing this offsetting for each of the subsequent groups of 15 rounds, enables a RAM machine to internally compute $Commit_L$ in its entirety in time linear in nL, and thus in λ.
- The output address of $Commit_L$ is $x = x_1, \ldots, x_{nL}$, where $x_i = 19(i-1)+15, \ldots, 19(i-1)+19$. Thus x can be internally computed and output in time linear in λ.
- The total number of identifiers involved in $Commit_L$ is $19nL$, and thus $O(nL \log nL) = O(v \log v) = O(\lambda)$ bits suffice to write them down. Thus a RAM machine can also output $Commit_L$ in linear in λ time.

Secondly, let us prove that algorithm $\mathbb{P}$ can be implemented so as to run in time linear in λ, and thus $\mathcal{P}_C$ and y can be constructed in time linear in λ. This is so because one can, in time linear in λ: (1) deduce the total number of nodes v as well as the number c of computational nodes and the number m of edges; (2) initialize one array A of v consecutive memory cells, the jth of which contains the incoming and outgoing edges of node j, and set aside an array B of m consecutive memory cells, the ith of which will contain a sequence of 5 integers —the label of edge i. Because each cell of these arrays can be accessed in constant time, it becomes trivial to verify that the entire internal computation of $\mathbb{P}$ can be completed in time linear in λ. (For instance, for each computation node j, $Func(j)$ is computable in constant time by retrieving —via array A— the j's incoming and outgoing edges and then counting them.) Further, outputting $(\mathcal{P}_C, y) = \mathbb{P}(C, x)$ can be done in time linear in λ. This can be argued as follows. By the previous remark, each AF^k of $\mathcal{P}_C$ is a constant-time RAM program that, on input the last 5 entries of the current public record R, generates a single action on at most 5 identifiers. Thus, each AF^k can be described by a number of bits upper-bounded by a constant times the maximum length of an identifier. Since each of our 4 basic computers has a constant number of rounds and generates a constant number of identifiers, the total number of rounds of $\mathcal{P}_C$ and the total number of identifiers generated are linear in c. Thus, the complete description of $\mathcal{P}_C$ and y is linear in $c \log c$, and thus linear in λ —which is $O(v \log v)$.

Thirdly, notice that the number of players N, the total number of rounds K, the player sequence PS, the start of outcome round J, and $Reveal_y$ are all trivially computable in linear time.

Finally, a RAM program for OF linear in λ can be described as a program that reads the last $5L$ entries of public record, divides them into L groups of 5 records, and then applies the $\mathbb{S}_5$ decoder to each group to obtain the desired L output bits.

Let now $\mathbb{C}$ be the algorithm that on input C, a λ-bit $E2$-representation of a standard normal-form mechanism $\mathcal{M} = (N, (\{0,1\}^L)^n, \{0,1\}^L, g)$, first computes $E3$-representations of $Commit_L$, $\mathcal{P}_C$, and $Reveal_L$ as specified above, and then concatenates them to obtain the desired perfect implementation $\mathcal{B} = (N, J, K, PS, AF, OF)$ of $\mathcal{M}$. Since such a concatenation too can be computed in time linear in λ, we have proved that $\mathbb{C}$ runs in linear time.

Let us now prove that $\mathcal{B}$ is linear-time by proving, in light of Explanation 4, that the number of computational steps required from each player is $O(\Lambda)$, where Λ is the length of $\mathcal{B}$'s $E3$-representation. This is so because each action function AF^k of $\mathcal{B}$ is a constant-time RAM program, and because we have just argued that OF is a RAM program running in time linear in L and thus in Λ.

Q.E.D.

G. Perfectly implementing any normal-form mechanism with private outcomes

Our results generalize to normal-form mechanisms $\mathcal{M}$ that, given a profile of messages $(m_1, \ldots, m_n)$, produce, not only a public outcome y, but also a private outcome y_i for each player i.

Intuitively, perfect implementations of such mechanisms can be obtained by a slight modification of our construction. First, consider a perfect ballot-box implementation $\mathcal{B}_P$ of the "public version" $\mathcal{M}_P$ of $\mathcal{M}$ that maps $(m_1, \ldots, m_n)$ to a public outcome with $n+1$ components: namely, $(y, y_1, \ldots, y_n)$. By the outcome-boundary round J, $\mathcal{B}_P$ thus produces $n+1$ sequences of envelopes, whose contents respectively encode $y, y_1, \ldots, y_n$ in a private and correct manner. Thus, to perfectly implement the original $\mathcal{M}$, one only needs to modify the ballot-box revealer of $\mathcal{B}_P$ so that (1) it publicly opens only the envelopes of the first sequence (thus making y publicly computable), and (2) it lets player i "privately read" the envelopes encoding y_i, thus enabling him to privately learn y_i.[41]

Let us now detail the changes to our definitions and construction necessary to formalize this intuition.

Changes to the definition of a normal-form mechanism. Definitions 1 is modified as follows:

- The outcome set Y is now the Cartesian product of $n+1$ subsets of Σ^*, $Y = Y_0 \times Y_1 \cdots Y_n$.

Definition 2 is restated as follows:

- A standard normal-form mechanism is a mechanism $\mathcal{M} = (N, M, Y, g)$ such that $M_1 = \cdots = M_n = Y_0 = \cdots = Y_n = \{0,1\}^L$ for some integer L.

[41] In sum, to implement normal-form mechanisms with private outputs, we just augment our ballot-box actions with the new action "Player i privately reads envelope j." While other augmentations are also possible, this one properly extends to the private-output setting the important "no residual information" property (i.e., Property 3) of perfect implementation. To illustrate this point, assume also including "hand envelope j to player i" in our set of ballot-box actions; and consider the following two implementations, $\mathcal{B}$ and $\mathcal{B}'$, of a secret referendum: namely the normal-form mechanism in which the players simultaneously choose votes and the outcome function returns the tally of their votes. Both $\mathcal{B}$ and $\mathcal{B}'$ privately and correctly produce n_1 sequences of envelopes, y containing the desired tallies, and y_i containing i's vote, and then, in the revealing stage, open all envelopes in y. However, $\mathcal{B}$ has player i privately read the envelopes in y_i, while $\mathcal{B}'$ hands over the envelopes in y_i to player i. It is immediately seen that $\mathcal{B}$ and $\mathcal{B}'$ are privately and strategically equivalent. Nonetheless, a player in $\mathcal{B}$ can only *claim* what his vote was, while a player in $\mathcal{B}'$ can actually *prove* what his vote was. Thus, buying votes is possible in a much stronger sense in $\mathcal{B}'$ than in $\mathcal{B}$, so that the two mechanisms may be played in a much different way!

CHANGES TO BALLOT-BOX ACTIONS. The following class of ballot-box actions is added:[42]

$(\textsc{PrivateRead}, j, i)$ — where i is a player and j is an envelope identifier in I_B.

"Player i privately reads the content of envelope j."

$R := R \circ (\textsc{PrivateRead}, j, i, \texttt{ub})$; $B = B \setminus \{B_j\}$; and $H_i := H_i \circ cont_B(j)$.

CHANGES TO THE DEFINITION OF A BALLOT-BOX MECHANISM. In essence, we continue to view a ballot-box mechanism for a normal form mechanism with private outputs as having a single output function OF, but let it have two inputs, and use the first one to indicate whether we are dealing with computing the public output or the private output of a player i. Formally, Definition 8, that is, the definition of an ℓ-bit ballot-box mechanism $\mathcal{B} = (N, K, J, PS, AF, OF)$, is modified as follows:

- Whenever k is greater than J, $AF^k(R)$ continues to be a set consisting of a single action, but this single action can now be either an open-envelope action or a private-read action.
- When the play of $\mathcal{B}$ terminates the outcomes are obtained as follows. The public outcome $y_0 \in \{0,1\}^\ell$ is obtained by evaluating function OF on inputs 0 and $Cont(R^{(J,K]})$. The private outcome $y_i \in \{0,1\}^\ell$ of player i is obtained by evaluating function OF on inputs i and $H_i^{(J,K]}$.
- In an execution e of $\mathcal{B}$ we distinguish the $n+1$ outcomes as follows: the public one is denoted by $out_0(e)$, and the outcome of player i by $out_i(e)$.

CHANGES TO THE DEFINITION OF A BALLOT-BOX REVEALER. Definition 14 is restated as follows:

- Let $y_0, \ldots, y_n$ be an address and $\mathcal{R} = (N, K, PS, AF)$ a ballot-box protocol. We say that $\mathcal{R}$ is a *ballot-box revealer* for addresses $y_0, \ldots, y_n$ if (1) each function in AF continues to return actions sets consisting of a single action, but this action can now either be an open-envelope or a private-read action; and (2) in any execution, envelope j is opened if and only if j belongs to y_0, and privately read by player i whenever j belongs to y_i.

CHANGES TO THE DEFINITION OF PERFECT IMPLEMENTATION. Definition 15 is modifed in items 5 and 6 as follow:

5. $J = K - (n+1)\ell L$; and
6. OF is the function that
   - 6.1 on input a pair consisting of 0 and $z = c_1, \ldots, c_{\ell L} \in \Sigma^{\ell L}$, applies the decoding function of $(\bar{0}, \bar{1})$ to z; and
   - 6.2 on input a pair consisting of $i > 0$ and a sequence $z_i \in \Sigma^{\ell L}$, applies the decoding function of $(\bar{0}, \bar{1})$ to z_i.

CHANGES TO THE DEFINITION OF STRATEGIC AND PRIVACY EQUIVALENCE. Definition 16 is restated as follows:

- Consider a general ballot-box mechanism $\mathcal{B} = (N, K, J, PS, AF, Y, OF)$ and a standard general mediated mechanism $\mathcal{M} = (N, (\{0,1\}^L)^n, (\{0,1\}^L)^{n+1}, g)$. We say that $\mathcal{B}$ is *general strategic and privacy equivalent* to $\mathcal{M}$ if there exists a probabilistic algorithm SIM_0 and $\forall i \in N$ there

[42]It may be natural to think of this action as three separate actions: (1) handing over an envelope to a player i; (2) having player i open the envelope and secretly read its contents; and (3) destroying the envelope after it has been privately read. Formalizing the first two actions would require the notion of a player "possessing" an envelope and thus would require re-defining a ballot to include the identity of the player possessing it. Although such formalization is possible, we find it more convenient to formalize this two-step process as a single ballot-box action.

exists a bijection $\psi_i : S_i^{\mathcal{B}} \leftrightarrow \{0,1\}^L$ and a probabilistic algorithm SIM_i such that, $\forall s \in S^{\mathcal{B}}$ the following properties hold:

Strategic Equivalence: $\langle (out(e), out_1(e), \ldots, out_n(e)) : e \leftarrow EX_{\mathcal{B}}(s) \rangle = g(\psi_1(s_1), \ldots, \psi_n(s_n))$.
Privacy Equivalence: $\langle\, (R^K(e), H_1(e), \ldots, H_n(e)) \,:\, e \leftarrow EX_{\mathcal{B}}(s) \,\rangle =$
$\langle\, F_0, F_1, \ldots, F_n \,:\, e \leftarrow EX_{\mathcal{B}}(s);\; F_0 \leftarrow SIM_0(out(e));\; \forall i \in N,\; F_i \leftarrow SIM_i(out_i(e), s_i) \,\rangle$.

CHANGES TO THE BALLOT-BOX CONSTRUCTIONS. Our construction of perfect implementation of mediated mechanisms stays the same, except for ballot-box revealers, that are constructed as follows:

- Let $y = y_0, \ldots, y_n$ be an address such that each y_i is a sequence of $5L$ envelope identifiers, $y_{i,1}, \ldots, y_{i,5L}$. Then the following is a ballot-box revealer for address y.

Protocol $Reveal_y$:

1. "For $\ell = 1$ to $5L$: publicly open envelope $y_{0,\ell}$."
2. "For $i = 1$ to n: For $\ell = 1$ to $5L$: "player i privately reads envelope $y_{i,\ell}$."

CHANGES TO THE PROOFS. All proofs are either syntactically identical, or totally analogous.

H. THE ISSUE OF ABORT

In an extensive-form mechanism, whenever it is player i's turn to act, the set of actions available to him must be fully specified. If $\mathcal{M}$ is a concrete mechanism and an action set of i is $\{Left, Right\}$, then it must indeed be the case that $Left$ and $Right$ are the only two options really available to him. In most concrete settings, however, player i will have the ability of taking no action at all.[43] Following computer science literature, we call such a "do-nothing action" an *abort*. Such an alternative option should not be confused with a non-participation option, which is an explicitly available strategic choice in many mechanisms. Because aborts are inherent in most concrete settings, any general theory of practical implementation must deal with them.

TECHNICALLY BYPASSING ABORT IN BALLOT-BOX MECHANISMS. The abort option does not arise in our ballot-box mechanisms, because they have been carefully engineered so as to *by-pass* this possibility. Indeed, whenever it is a player's turn to act, his action set in our ballot-box mechanisms consists of (1) a single public action or (2) two complementary private actions. In the first case, the action can be performed by *anyone*: all that the players need to see is that the action has been performed. In particular, public actions can be performed by an external agent, even one not trusted by the players, making the abort issue disappear completely. In the second case, while the other players "look away," the acting player needs either to swap two ballots or leave them alone. Thus, performing no action is formally equivalent to leaving them alone.

[43] Assume that, in a concrete mechanism, the acting player i is at an intersection, and that his action set is $\{(turn)\ Left, (turn)\ Right\}$. Then, for him to really have no other action available, he must be driving a car with no breaks, on a road very steeply downhill, etc. Else, he would always have the option of standing still rather than moving in either direction.

ALTERNATIVE WAYS OF HANDLING ABORTS. One alternative way to handling the abort issue consists of having the mechanism designer (1) formally include *abort* as an action available in each action set of a concrete mechanism; (2) assign special *abort outcomes* to terminal nodes reached after an abort by some player; and (3) provide incentives to prevent players from aborting. Strategic equivalence can thus be formulated as outcome-preserving bijections between the players' strategies in the normal-form mechanism and their non-aborting strategies in its concrete implementation. In fact, with proper incentives, all strategies that lead to aborting outcomes can be iteratively eliminated as being dominated.[44] This alternative was indeed taken in Izmalkov, Lepinski, and Micali (2005) assuming that, as soon as a player played the abort action, a special terminal node is immediately reached and a sufficiently high fine imposed on the aborting player.

REFERENCES

AUMANN, R. J., AND S. HART (2003): "Long Cheap Talk," *Econometrica*, 71(6), 1619–1660.

BÁRÁNY, I. (1992): "Fair Distribution Protocols or How the Players Replace Fortune," *Mathematics of Operations Research*, 17, 329–340.

BARRINGTON, D. A. (1986): "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1," in *Proceedings of the 18th Symposium on Theory of Computing*, pp. 1–5. ACM.

BEN-OR, M., S. GOLDWASSER, AND A. WIGDERSON (1988): "Completeness theorems for fault-tolerant distributed computing," in *Proceedings of the 20th Symposium on Theory of Computing*, pp. 1–10. ACM.

BEN-PORATH, E. (1998): "Correlation without Mediation: Expanding the Set of Equilibrium Outcomes by Cheap Pre-Play Procedures," *Journal of Economic Theory*, 80, 108–122.

——— (2003): "Cheap talk in games with incomplete information," *Journal of Economic Theory*, 108(1), 45–71.

BOLTON, P., AND M. DEWATRIPONT (2005): *Contract Theory*. MIT Press, Cambridge, Massachusetts.

BRANDT, F., AND T. SANDHOLM (2004): "(Im)possibility of unconditionally privacy-preserving auctions," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 810–817. IEEE.

CHAUM, D., C. CRÉPEAU, AND I. DAMGÅRD (1988): "Multi-party unconditionally secure protocols," in *Proceedings of the 20th Symposium on Theory of Computing*, pp. 11–19. ACM.

CLEVE, R. (1986): "Limits on the Security of Coin Flips when Half the Processors are Faulty," in *Proceedings of the 18th Symposium on Theory of Computing*, pp. 264–369. ACM.

[44]In a second alternative way, the normal-form mechanism may be properly augmented *too*. That is, *abort* is formally added as a distinguished element to the message set of each player, distinguished "abort outcomes" are formally added to the outcome set, and the outcome function is properly extended so as to keep the current formulation of strategic equivalence intact.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001): *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts.

Cramton, P., and J. Schwartz (2002): "Collusive Bidding in the FCC Spectrum Auctions," *Contributions to Economic Analysis & Policy*, 1(1), 1078–1078.

Dodis, Y., S. Halevi, and T. Rabin (2000): "A cryptographic solution to a game theoretic problem," in *In Advances in Cryptology — CRYPTO 2000, LNCS*, vol. 1880, pp. 112–130. Springer-Verlag.

Forges, F. (1990): "Universal Mechanisms," *Econometrica*, 58, 1341–1364.

Gerardi, D. (2004): "Unmediated communication in games with complete and incomplete information," *Journal of Economic Theory*, 114(1), 104–131.

Gerardi, D., and R. Myerson (2005): "Sequential Equilibria in Bayesian Games with Communication," Yale University, mimeo.

Goldreich, O., S. Micali, and A. Wigderson (1987): "How to play any mental game," in *Proceedings of the 19th Symposium on Theory of Computing*, pp. 218–229. ACM.

Goldwasser, S., S. Micali, and C. Rackoff (1985): "The knowledge complexity of interactive proof-systems," in *Proceedings of the 17th Symposium on Theory of Computing*, pp. 291–304. ACM, Final version in *SIAM Journal on Computing*, 1989, 186–208.

Gordon, S. D., and J. Katz (2006): "Rational Secret Sharing, Revisited," Cryptology ePrint Archive, Report 2006/142, `http://eprint.iacr.org/`.

Grimm, V., F. Riedel, and E. Wolfstetter (2003): "Low price equilibrium in multi-unit auctions: the GSM spectrum auction in Germany," *International Journal of Industrial Organization*, 21(10), 1557–1569.

Halpern, J. Y., and V. Teague (2004): "Rational secret sharing and multiparty computation," in *Proceedings of the 36th Symposium on Theory of Computing*, pp. 623–632. ACM.

Hopper, N., J. Langford, and L. a. von Ahn (2002): "Provably Secure Steganography," in *Proceedings of the Crypto 2006.*

Izmalkov, S., M. Lepinski, and S. Micali (2005): "Rational secure computation and ideal mechanism design," in *Proceedings of the 46th Symposium on Foundations of Computer Science*, pp. 585–594. IEEE.

Krishna, R. V. (2006): "Communication in Games of Incomplete Information: Two Players," *Journal of Economic Theory*, forthcoming.

Krishna, V. (2002): *Auction Theory.* Academic Press, San Diego, California.

Lepinski, M., S. Micali, C. Peikert, and A. Shelat (2004): "Completely fair SFE and coalition-safe cheap talk," in *Proceedings of the 23rd annual Symposium on Principles of distributed computing*, pp. 1–10. ACM.

Lepinski, M., S. Micali, and A. Shelat (2005): "Collusion-Free Protocols," in *Proceedings of the 37th Symposium on Theory of Computing*, pp. 543–552. ACM.

Lindell, Y., and B. Pinkas (2004): "A Proof of Yao's Protocol for Secure Two-Party Computation," available at: `http://www.cs.biu.ac.il/~lindell/abstracts/yao_abs.html`.

Lysyanskaya, A., and N. Triandopoulos (2006): "Rationality and Adversarial Behavior in Multi-Party Computation," in *Proceedings of the Crypto 2006*.

Naor, M., B. Pinkas, and R. Sumner (1999): "Privacy Preserving Auctions and Mechanism Design," in *Proceedings of the 1st conference on Electronic Commerce*. ACM.

Papadimitriou, C. H. (1993): *Computational Complexity*. Addison Wesley, Boston, Massachusetts.

Posner, R. A. (1981): "The Economics of Privacy," *The American Economic Review*, 71(2), 405–409, Papers and Proceedings of the Ninety-Third Annual Meeting of the American Economic Association.

Rabin, T., and M. Ben-Or (1989): "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority," in *Proceedings of the 21st Symposium on Theory of Computing*, pp. 73–85. ACM.

Rothkopf, M. H., T. J. Teisberg, and E. P. Kahn (1990): "Why are Vickrey Auctions Rare?," *Journal of Political Economy*, 98, 94–109.

Saijo, T., T. N. Cason, and T. Sjöström (2003): "Secure Implementation Experiments: Do Strategy-proof Mechanisms Really Work?," Discussion papers 03012, Research Institute of Economy, Trade and Industry (RIETI), available at http://ideas.repec.org/p/eti/dpaper/03012.html.

Sjöström, T., and E. Maskin (2002): *Handbook of Social Choice and Welfare*vol. 1, chap. Implementation Theory, pp. 237–288. North-Holland.

Stigler, G. J. (1980): "An Introduction to Privacy in Economics and Politics," *The Journal of Legal Studies*, 9(4), 623–644, The Law and Economics of Privacy.

Urbano, A., and J. E. Vila (2002): "Computational Complexity and Communication: Coordination in Two-Player Games," *Econometrica*, 70(5), 1893–1927.

Wilson, R. (1987): "Game-theoretic Analyses of Trading Processes," in *Advances in Economic Theory, Fifth World Congress*, ed. by T. F. Bewley, pp. 33–70. Cambridge University Press, Cambridge, UK.

Yao, A. (1986): "Protocol for Secure Two-Party Computation," never published. The result is presented in Lindell and Pinkas (2004).