

Perfect Implementation of Normal-Form Mechanisms

By Sergei Izmalkov, Matt Lepinski, and Silvio Micali*

October 21, 2008

Abstract

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 mediator or violating the privacy of the players*.

We prove that *any* normal-form mechanism can be perfectly implemented by a *public mediator* using envelopes and an envelope-randomizing device (i.e., the same tools used for running fair lotteries or tallying secret votes). Differently from a trusted mediator, a public one only performs prescribed public actions, so that everyone can verify that he is acting properly, and never learns any information that should remain private.

1 Introduction

A game consists of a *context*—describing the outcomes, the players (including their types and beliefs), and the players’ preferences over the outcomes— and a *mechanism* (or a game form), describing the actions available to the players, and the way in which actions lead to outcomes. The usual goal of mechanism design consists of finding a mechanism that, for a given class of contexts, defines a game whose equilibria yield a desired outcome. While contexts can be arbitrarily complex, mechanism design strives to find simple mechanisms. A *normal-form* mechanism consists of a set of *reports* (or messages) for each player, and an *outcome function* mapping these reports to outcomes.

Normal-form mechanisms are designed so as to enjoy valuable theoretical properties. As defined, however, they are *abstractions*. Outcome functions do not spontaneously evaluate themselves on players’ reports. To be useful in real strategic settings, normal-form mechanisms are concretely implemented with the help of a *mediator*. But such concrete implementations present theoretical difficulties that are far from being understood. Indeed, we argue that

Unless they satisfy several new goals, not achieved and often not addressed so far, concrete mediated implementations of normal-form mechanisms not only are not very meaningful, but fail to enjoy the very properties achieved by their abstract counterparts.

We contend that without a rigorous and sufficiently powerful theory of concrete implementation, the practical meaningfulness of normal-form mechanisms is at risk. Providing such a theory is the goal of this paper.

1.1 Our Goals for Mediated Mechanisms

Our goals for mediated mechanisms are very stringent. They pertain to *privacy and trust, physical assumptions, operational efficiency, neutrality to collusion, and constructiveness*. Proceeding informally, this introduction first illustrates our goals, and then explains in what sense they can be simultaneously achieved.

*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.

1.1.1 Privacy and Trust

Among auctions in the private-value setting, the famous second-price sealed-bid mechanism is a normal-form mechanism whose reports consist of numerical bids—one for each player—and whose outcome function returns the player with the highest bid as the *winner*, and the value of the second-highest bid as the *price*. The characteristic property of this mechanism is *economic efficiency achieved in dominant strategies*. Indeed, because it is optimal for each player to report his true valuation no matter what the others may report, the player with the highest valuation wins the item for sale. We shall use this mechanism to illustrate the problems of privacy and trust when abstract normal-form mechanisms are concretely implemented with the help of a mediator.

Private vs. Public Mediators. Consider the following two mediated implementations of the second-price sealed-bid mechanism.

M_1 (*With a Private Mediator*): The players seal their bids into envelopes and hand them to the mediator, who then privately opens them, privately evaluates the outcome function, and finally publicly announces the winner and the price.

M_2 (*With a Public Mediator*): The players seal their bids into envelopes and hand them to the mediator, who then publicly opens all of them, so as to enable everyone to compute the winner and the price.

Notice that the mediators of these two implementations are asked to perform different types of actions. The mediator of implementation M_2 only performs *public actions*, so that everyone can verify that the right actions have been performed. By contrast, the mediator of M_1 is asked to perform *private actions*, so that only he knows whether he has performed the right actions. Accordingly, these two implementations are at opposite ends with respect to privacy and trust. On the one extreme, implementation M_1 reveals nothing more than the announced outcome, but requires total trust in the mediator. Indeed, the players cannot verify whether the mediator announces the correct outcome, nor whether he will keep all bids secret after the auction is over.¹ On the other extreme, M_2 guarantees the correctness of the outcome, but makes the entire players' reports public knowledge. In sum, M_1 requires total trust and offers total privacy, while M_2 requires no trust and offers no privacy.

To be sure, a public mediator is still trusted, but in a different, “public” sense: he is trusted to really perform whatever public actions he is asked to perform. For instance, in implementation M_2 he is trusted not to open just the first half of the envelopes and destroy the other half. Clearly such trust is much milder: because any deviation from prescribed public actions is immediately observed, a public mediator can be kept accountable.

Privacy and Trust as Strategic Problems. Privacy-valuing players may avoid participating in implementation M_2 , and distrustful players in implementation M_1 . Such reduced participation not only causes the seller to fetch lower prices, but negatively affects economic 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 economically 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 economically efficient, even with respect to just the participating players*. Let us explain.

- 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.² Such a player will not find it optimal to report his true valuation to the mediator;

¹According to Rothkopf, Teisberg and Kahn (1990), such concerns explain why second-price auctions are rarely used in practice.

²If 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

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, concrete implementation M_1 may not be 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 make concrete implementation M_2 inefficient as well.

More generally, privacy and trust affect the players’ strategic thinking, making it unclear whether mediated implementations 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 economic 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 efficient in the “revised” valuations, while Society’s interest is that it be economically efficient in the “original” valuations. Integrating privacy and strategic concerns in the design of concrete implementations of normal-form mechanisms is a *new* problem, and requires new techniques and conceptual frameworks.

Our Goal. At a closer look, 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:

A concrete implementation of the second-price mechanism should always enjoy economic efficiency (or any other desired property) to exactly the same degree (whatever it may be) as the abstract mechanism.

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. That is:

A meaningful mediated implementation of a normal-form mechanism \mathcal{M} should preserve exactly all of \mathcal{M} ’s theoretical properties, including its privacy properties, without trusting the mediator at all.

1.1.2 Physical Assumptions

Concrete implementations, by definition, ultimately “live” in our physical world, and thus Nature has something to say on whether sufficiently good mediated implementations exist. (At a minimum, one must assume the existence of a physical way —such an opaque envelope— enabling the players to privately transfer their messages to the mediator.) Accordingly, nothing can be *rigorously proved* about concrete implementations of normal-form mechanisms without making *explicit assumptions* about Nature.³ Theorems require axioms, and theorems about physical implementation require physical axioms. But after realizing that physical assumptions are necessary, we must resist the temptation of assuming anything we want, without any concern about reasonableness.

bribe for his disclosure).

³ Note that we speak about physical “assumptions” rather than “laws” because we are interested in formal proofs rather than experiments, and from a mathematical point of view any current physical law (whose “truthfulness” depends on the current body of experiments) is an assumption about Nature. Indeed, physical laws are refined over time —to say the least.

A Common Temptation. An easy temptation in implementing normal-form mechanisms is to replace their mediators with computers, de facto damping all difficulties about privacy and trust on Nature’s shoulders. That is, rather than providing his own message m_i to the mediator, every player i privately inputs m_i to a computer. After that, the computer runs (the software for) the outcome function on the received inputs and then publicly outputs the desired result. Although conceptually very simple, this approach is also very naive. Indeed, it is trivial to manufacture computers so as to hide a functionality different from the declared one. So: whose computer should the players choose? (“If you do not mind, let it be mine!”) In addition, as for all electronic devices, computers’ functionality can be electronically altered “from afar.” Accordingly, a computer’s *correctness*—especially in the presence of malicious external influences—cannot be too easily guaranteed by realistic physical assumptions and “inspection procedures.” But even taking for granted that correctness could be guaranteed, a lot of “cooperation” would still be required from Nature in order to guarantee a computer’s *privacy*. Computers are in fact notorious to leak information about the data they operate upon in unlimited (and beforehand unpredictable) ways. For instance, Kocher, Jaffe and June (1999) show how cryptographic keys, stored within a computer and typically consisting of a few thousand bits, can be easily reconstructed from monitoring the amperage used during a cryptographic computation. This is so because processing a 1 is a more complex operation—and thus absorbs “more electricity”—than processing a 0. Thus, in our envisaged application, a player with a reasonably good amperometer connected to the outlet to which the trusted computer is plugged into, can get excellent information about the reports of other players. To prevent this loss of privacy, one may require that the envisaged computer works on batteries instead. Even so, the electro-magnetic radiation alone (even measured by very rudimentary devices) has been showed to be totally sufficient in the reconstruction of cryptographic keys—e.g., see Agrawal, Archambeault, Rao, and Rohatgi (2002). What’s next? The danger is that every “fix” may have a new vulnerability.

In sum, computers are just the wrong way to go when looking for meaningful and concrete implementations. If one aims to *prove* that they can be relied upon to remove any trust from mediators, while *provably* guaranteeing total privacy, he must be prepared to explicitly state extraordinary assumptions about Nature.

Our Goal. While some explicit assumptions about Nature are necessary to prove that sufficiently good mediated implementations of normal-form mechanisms exist, one should *not* simply assume that “if mediators cannot guarantee privacy and trust, then Nature will.” Accordingly,

Implementations of normal-form mechanisms should be based on few, simple, and reasonable physical assumptions.

1.1.3 Operational Efficiency

Any mechanism needs resources to be executed. An abstract mechanism solely requires *computational* resources, as dictated by the complexity of its outcome function. A concrete mechanism, because it is executed in the physical world, also requires *physical resources*. Clearly, concrete mechanisms cannot use less resources than the abstract mechanisms they implement, but neither can they be deemed meaningful if they use so many more resources that they cannot be implemented in our physical universe.

Focusing on finite mechanisms, we consider only outcome functions with finite domain and range. In computer science, the traditional way to measure the complexity of a finite function f is the number of elementary computational operations that must be performed to evaluate f on any possible input. A typical such operation is AND: the function mapping any two bits to 1 if they are equal to each other, and to 0 otherwise.

A Common Source of Operational Inefficiency. Concrete implementations are often operationally inefficient because they conceptualize finite functions as *tables* rather than *algorithms*. That is, an outcome function g is treated as an ordered list—indexed by all possible inputs x —of all possible outputs $g(x)$. In this fashion, all functions have no structure at all, and thus all are equally simple to describe, but also equally inefficient to implement. For instance, consider the following mediated implementation of a two-

player normal-form mechanism, whose outcome function g has k -bit inputs. (The implementation aims to preserve the privacy of the players’ reports.)

Concrete Implementation *Naive*

1. A public mediator generates a $2^k \times 2^k$ matrix of envelopes as follows. For each possible pair of reports (m_i, m_j) , a mediator publicly seals $g(m_i, m_j)$ into a separate envelope, and makes it the (i, j) item of the matrix. The mediator then hands the entire “envelope matrix” to Player 1.
2. Player 1 secretly permutes the rows, so that the j th row of envelopes becomes the first row if his secret report is j , and hands the permuted matrix to Player 2.
3. Player 2 secretly permutes the columns, so that his intended report corresponds to the first column, and hands the further permuted matrix to the mediator.
4. The mediator finally opens the top-left corner envelope of the matrix to reveal the outcome.

We do not consider *Naive* a meaningful implementation, because its extreme simplicity is accompanied by its extreme operational inefficiency.⁴ Indeed, even when the players’ reports consists of just 150 bits (less than those necessary to described the players’ “types” in many practical contexts), *Naive* requires a number of envelopes and a number of physical operations (i.e., “stuffing” and permuting envelopes) greater than the total number of elementary particles. Informally, therefore, the universe does not have enough “matter” to manufacture *Naive*’s required envelopes.

Can an implementation requiring more resources than those available in the Universe be deemed concrete? Perhaps yes, if the outcome function g itself had complexity exponential in k . (In this case, in fact, it could not be feasibly evaluated, independently of any considerations about privacy, trust, etc.) But absolutely NO, if g ’s complexity were polynomial in k . The latter is very often the case. For instance, putting aside the fact the reports are typically short in single-good auctions, the complexity of the second-price outcome function is actually linear in k .⁵

Remark. As we shall discuss, the entire literature on pre-play communication treats functions as tables. Accordingly, all their concrete protocols are totally inefficient.

Our Goal. Without ruling out *Naive* and similar approaches, any theory of concrete implementation will remain just “theory.” Accordingly, we should treat outcome functions as algorithms rather than tables, and demand that

If the outcome function of a normal-form mechanism \mathcal{M} has complexity k , a meaningful implementation of \mathcal{M} should have complexity close to k .

1.1.4 Neutrality to Collusion

The properties of a mechanism must hold *at equilibrium*, and equilibria are typically defined with respect to “single-player deviations”. Accordingly, a mechanism might very well be vulnerable to collusion. That is, its desired properties may no longer hold if two or more players deviate in a coordinated manner from their equilibrium strategies.

Letting \mathcal{M}' be a concrete implementation of a normal-form mechanism \mathcal{M} , it is clear that \mathcal{M}' cannot be less vulnerable to collusion than \mathcal{M} . However, vulnerability to collusion may be a small problem for \mathcal{M} (because its outcome function provides only small incentives to colluding players, and thus no collusion may actually occur), and instead be a huge problem for \mathcal{M}' . This phenomenon is actually quite common for the concrete implementations proposed in the literature so far. And it should be avoided at all costs, because

⁴In addition, without some crucial additions and changes, it risks of not being private —see the discussion of logical structure of Section 1.3.

⁵In fact, it only needs to compare the bits of the two reported bids one at a time, starting with the most significant one, until it finds a bit-position for which one bids has 1 and the other has 0.

coalitions will surely form when executing \mathcal{M}' , causing its plays to be vastly different from those of \mathcal{M} . To illustrate this phenomenon in an elementary way, consider the following “extreme” implementation of the second-price auction mechanism (SPA for short).

Concrete Implementation *Extreme SPA*

In a first step, all players simultaneously and privately submit their bids to players 1 and 2, who wait for such bids in the same private room. In a second step, players 1 and 2 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.

Notice that implementation *Extreme SPA* maintains the economically efficient equilibrium of SPA —although not in dominant strategies. Notice too, however, that SPA and *Extreme SPA* are quite different in their vulnerability to collusion. In SPA, no pair of players can “control the outcome.” (For instance, no matter how two collusive players might behave, the revenue generated by SPA is always greater than or equal to the second-highest bid of the “independent” players.) Quite differently, in *Extreme SPA*, players 1 and 2 *fully and undetectably control the outcome*, that is, they can choose —without fear of being discovered and without recourse— any winner and price they want. In particular, based on the bids received in the first step, they can implement the second-price auction only “between the two of them.” That is, they can announce the winner to be the one of them with the highest bid, and the bid of the other as the price —which is individually incentive compatible. In other words, while SPA does not provide much incentives for a coalition of two players to form, in *Extreme SPA* players 1 and 2 have plenty of motivation and indeed the means to collude. Accordingly, quite differently from SPA, we should expect players 1 and 2 to collude in *Extreme SPA* and the revenue generated to be negligible. Notice that the real problem here is allowing the outcome to be controlled by just two players, not in providing a private room for these two players. Indeed, being rational, players 1 and 2 are aware of their power, and we should expect that they will find a way to exercise it in a purely non-cooperative way.

Remark. The entire pre-play communication literature empowers a subset of the players (ranging from just two players to half of the players) to totally and undetectably control the outcome. Accordingly, if one were to use pre-play protocols as concrete implementations, it would be highly questionable whether their actual plays would resemble those of their abstract counter-parts.

Our Goal. True to its name, *Extreme SPA* provides a subset of the players such extraordinary powers that it is hard to consider it a meaningful implementation of the second-price mechanism. But even giving a set of players a moderate *additional* power (i.e., a power they do not have in the original abstract mechanism) should not be easily condoned in a concrete implementation. Ideally, we should demand that

A concrete implementation of a normal-form mechanism should not provide any additional power (or information) to any subset of players.

1.1.5 Constructiveness and Context Independence

In line with the *ontological argument*, a necessary pre-requisite of a perfect implementation is its existence! However, even a theorem guaranteeing that “perfect” implementations always exist might not have any potential relevance to practice. If its proof were totally non-constructive, we would not know where to start looking for the perfect implementations we need. But constructiveness in its most generic sense —namely, the ability of finding perfect implementations by means of an automatic procedure— might not be satisfactory either. Should such a procedure take astronomical time, we would not live long enough to see its results. Therefore, *perfect implementations not only should be efficient to play, but also efficient to find.*

The Wilson’s Doctrine for Mechanism Design. Even an efficient procedure to construct perfect implementations would be of scarce potential applicability if it did not have a proper “structure”. Let us explain. 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 obtaining perfect implementations might very well consist of an algorithm that, on input *both* a normal-form mechanism \mathcal{M} and a context \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. We instead want a procedure that is *context-free*. Namely, a procedure that, on input an arbitrary normal-form mechanism \mathcal{M} , generates a concrete mechanism \mathcal{M}' such that, for all possible contexts \mathcal{C} , the players are totally indifferent between playing the games $(\mathcal{M}, \mathcal{C})$ and $(\mathcal{M}', \mathcal{C})$. In essence, we want to achieve for mechanism implementation what Wilson (1987) advocates for mechanism design: namely, *to find solutions minimizing dependence on the details of the problems at hand* (the “Wilson’s Doctrine”).

Let us now put together the two desiderata of this sub-section.

Goal. *We want a universal and efficient procedure that, given just the specification of an abstract normal-form mechanism, quickly specifies an equivalent and perfect implementation of it in any possible context.*

1.2 Our Results

Our contribution consists of rigorously defining the above goals and then showing that *all* of them can be *simultaneously* achieved. In essence, we prove that

Any finite normal-form mechanism (with any number of players) has a perfect implementation, based on the ability of randomizing perfect ballots, which is both easy to construct and easy to play.

Let us now proceed to informally explain the above statement.

Our Notion of a Perfect Implementation. Informally, we say that an extensive-form mediated mechanism \mathcal{M}' *perfectly implements* a normal-form mechanism \mathcal{M} if the following properties hold:

No Personal Trust: The mediator of \mathcal{M}' is public —i.e., only performs a publicly and uniquely specified sequence of public actions.

Strategy Equivalence: There is a one-to-one correspondence between each player’s strategies in \mathcal{M} and his strategies in \mathcal{M}' such that the outcome of any profile of strategies in \mathcal{M} is identical to the outcome of the profile of the corresponding strategies in \mathcal{M}' .

Privacy Equivalence: No set of players gains more information (about the other players’ strategies) in \mathcal{M}' than they do in \mathcal{M} . In essence, except for the outcome, all information divulged consists of random noise.

Consequently, from a strategy/privacy perspective, individual players as well as arbitrary subsets of the players are indifferent, for any context \mathcal{C} , between playing the games $G = (\mathcal{C}, \mathcal{M})$ and $G' = (\mathcal{C}, \mathcal{M}')$. In particular, the two games enjoy identical solutions. Indeed, although \mathcal{M}' is extensive-form, strategy equivalence states that \mathcal{M} and \mathcal{M}' have identical —up to renaming and reordering of strategies— normal-form representations. Of course, all 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. In particular, therefore, perfect implementations preserve all traditional equilibria and set-valued solutions.

Our Physical Assumptions. Our construction relies on two physical assumptions, summarized as follows.

- *Perfect Ballots.* We essentially rely on ballots of two different sizes, envelopes and super-envelopes, satisfying the following properties: (1) any two ballots of the same size are identical from the outside, (2) any ballot provides no information about its content, and (3) any ballot guarantees the integrity of its content until it is opened.
- *Perfect Ballot-Box.* We essentially rely on a bingo-like machine, the ballot-box, that randomly and secretly permutes a sequence of ballots of the same size. That is, given k ballots, the ballot-box returns with equal probability one of the $k!$ permutations of these ballots, without revealing any information about the actual permutation.

In essence, therefore, we rely on the same “hardware” used for running fair lotteries and tallying secret votes *in the real world* and from time immemorial. We thus use classical primitives to achieve new desiderata.

Our Operational Efficiency. Our concrete implementations are as efficient as theoretically possible. Namely, if a normal-form mechanism has an outcome function of complexity k , then it also has a perfect implementation whose complexity is *linear* in k . Informally, we define the complexity of a perfect implementation to consist of the total number of its elementary steps, whether computational or physical. For our implementations, an elementary physical step consists of a simple operation on a set of ballots —such as publicly opening a ballot to reveal its content to everyone, or “ballot-boxing” at most five ballots.

To achieve this operational efficiency our implementations take advantage and indeed closely mirror the algorithmic structure of the relevant outcome functions g . That is, if g consists of k elementary computing steps, then a perfect implementation simulates each such step individually, by means of a constant number of public and elementary operations on ballots.

Specifically, in our implementations each bit corresponds to a sequence of 5 envelopes, whose contents indeed encode the bit in question in a special way; and each elementary computational step corresponds to a *fixed* number of elementary and public ballots operations. For instance, if a sequence of 5 envelopes encodes a bit a and a disjoint sequence a bit b , then the prescribed sequence of public operations corresponding to an AND transforms these ten envelopes into a new sequence of 5 envelopes, whose content is guaranteed to contain an encoding of the bit $c = \text{AND}(a, b)$, without revealing any information whatsoever about a or b . Thus, if g consists of k elementary computational steps, then our “enveloped computation” of g consists of ck elementary physical operations, where c is a number of ballot operations sufficient to implement each of the elementary computational steps of g . (As we shall see, any finite function can be evaluated with just 4 types of elementary computational steps.)

Our Neutrality to Collusion. Note that, unlike for *Extreme SPA*, a perfect implementation \mathcal{M}' of a normal-form mechanism \mathcal{M} guarantees that no *additional* collusive power exists. Indeed, strategy equivalence and privacy equivalence guarantee that not only any individual player but also *any subset* of players (including the set of all players) have *identical strategic opportunities and information* in a play of \mathcal{M} and in a play of its perfect implementation \mathcal{M}' . Therefore, perfect implementations provide no additional incentives for any coalition of players —of any size— to form. (Such bounding of collusive power was first studied, in a slightly weaker sense, by Lepinski, Micali, Peikert and Shelat (2004).)

Our Constructiveness and Context Independence. The automatic and efficient constructibility of our perfect implementations is an immediate corollary of the fact that we implement each elementary computational step, in isolation, via a fixed number of public and elementary operations on ballots. Thus we can immediately translate any convenient sequence of elementary computational steps to compute an outcome function g into a constant-factor longer sequence of public ballot operations.

Moreover, since we directly translate the outcome function of the normal-form mechanism and not the context in which the abstract mechanism is envisaged to be played, our perfect implementations are automatically context-free.

1.3 Additional Aspects of Our Results

Logical Structure. Our perfect implementations of normal-form mechanisms work in three stages. In the *input stage* each player simultaneously hands over to the public mediator a sequence of envelopes whose contents encode—in a special way—his intended report. In the *computation stage*, the public mediator performs a sequence of public operations on the received sequences of envelopes so as to obtain a final sequence of envelopes that is guaranteed to encode the correct outcome. In the *output stage*, the public mediator opens all final envelopes so as to enable everyone to find what the outcome is. Thus:

Perfect implementations have the same logical structure of traditional mediated ones.

That is, the players touch only the envelopes containing their respective secret reports, hand them to the mediator at the very start of a play, and take no further actions. Let us point out some immediate advantages of this structure.

- *No Privacy Risks.* In contrast with our logical structure, recall that implementation *Naive* of Section 1.1.3 entrusts player 2 with the sequence of envelopes encoding the secret report of player 1, thus putting player 1’s privacy at risk. For instance, it would be easy for player 2 to run away with player 1’s envelopes. But since his doing so would be publicly observed, a possible way to prevent this loss of privacy might consist of explicitly modifying *Naive* so that a suitable *fine* is imposed on player 2 if he so misbehaves. For such a fine to be an effective deterrent, however, the mechanism designer should know beforehand an upperbound for the value to player 2 of learning player 1’s report. And such knowledge is not always available to the designer: in mechanism design in its purest form, it resides with the players. But even if the context at hand made it possible to find a sufficient fine, in a play of *Naive* player 2 might instead (without being caught) have a peak into player 1’s envelopes when he is given the privacy necessary to permute the columns of the envelope matrix. Perhaps these privacy problems could be fixed by appropriate changes to *Naive*—although the value of fixing a totally inefficient solution can be questioned. But it is certainly better to insist on a logical structure in which no player handles the reports of the others in any way.

To appreciate this point, consider Israelis and Palestinians implementing a negotiating mechanism. Then, our construction asks Israelis and Palestinians to hand envelopes encoding their respective negotiating positions to the UN security council, so that the latter can perform on these envelopes a sequence of public actions proven to yield the desired negotiated outcome without unnecessarily revealing the initial positions. By contrast, implementation *Naive* asks the Israelis (or the Palestinians) to hand envelopes encoding their position to the other player, so that they can perform a secret operation on them. Not quite the same thing.

- *No Strategic Signaling.* It is well known that the ability of signaling significantly alters the way mechanisms are played. Being extensive-form mechanisms, perfect implementations are in principle affectable by signaling. However, our logical structure prevents this from happening. In fact, the players simultaneously act at the very first round, and thus without signaling to each other. At that point, *les jeux sont faits*: their reports are already “committed to” and cannot be altered, and only the verifiable mediator acts from then on.
- *No Random Signaling.* It is also well known that even random signals, outside of the players’ control, can alter the way games and mechanisms are played. And by divulging random noise (before revealing the final outcome), perfect implementations in principle could be affected by random signals. It is again our logical structure that prevents this alteration of play. Because a perfect implementation starts with the players simultaneously handing their envelopes to the mediator, no random signal is available to them when they choose their reports once and for all. After that, the prescribed actions of the mediator reveal truly random noise, but by then it is too late for the players to act on such common random signals.

Private Outcomes. So far, we have discussed normal-form mechanisms just with *public outcome functions*, that is, mechanisms whose outcomes are publicly revealed. More generally, in addition to the public outcome, a mechanism may have a *private outcome* for each player; that is, it may reveal to each player a “customized piece of information.” For instance, one can envisage a second-price mechanism to be defined so as only the winner (and the seller) learns that she has won and the price she has to pay, while all other bidders learn only that they did not win.

We construct perfect implementations for such general mechanisms as well. At a high level, each player continues to hand to the mediator a sequence of envelopes encoding his own secret report, and the mediator continues to perform on these envelopes a uniquely defined sequence of public actions. But now, letting n be the number of players, these public actions produce $n + 1$ sequences of envelopes: the j th sequence encodes the private outcome of player j , and the last sequence encodes the public outcome. Then, the mediator publicly opens the last sequence and publicly hands over the j th sequence to player j .

Handling Aborts. In an abstract normal-form mechanism, each player can only choose a report from his specified message set. However, even when running the mechanism with the help of an *angel* (so that there is no issue about trust) any player has the option of *aborting*, that is, handing to the angel an envelope containing something outside his message set —e.g., an insult— or nothing altogether. In this case, the angel has to be provided with precise instructions on what to do. Else, ultimately, the angelically mediated game is not fully specified and cannot be properly analyzed.

It should be realized that, in any concrete setting, there are infinitely many ways of aborting. For instance, consider an implementation of the second-price auction with a public mediator in which (a) each player — simultaneously with the others— is supposed to deliver to the mediator a sealed envelope containing a 20-bit bid, and (b) the mediator is supposed to publicly open all received envelopes. Then, what if a player i instead delivers a cat or an envelope containing a drawing? In both cases, everyone will realize that player i has aborted, either immediately or after all envelopes are publicly opened, but what should the mediator do? A natural way to complete the specification of this or any other publicly mediated normal-form mechanism consists of (1) treating all possible “illegitimate” actions as the single strategy **abort** and (2) treating an aborting player as having publicly submitted a specific message —e.g., the bid 0 in our auction example.⁶

Technically, the original normal-form mechanism \mathcal{M} is augmented so that (1′) the distinguished strategy **abort** is added to the original strategy set M_i of each player i ; and (2′) the input **abort** from player i is replaced with a specific message $\overline{m}_i \in M_i$ when evaluating the original outcome function. For an extensive-form, publicly mediated mechanism \mathcal{M}' to be a perfect implementation of the so augmented \mathcal{M} , strategy equivalence and privacy equivalence must be modified so as to account for aborting strategies. First, it must be guaranteed that the set of strategies of each player i in \mathcal{M}' can be decomposed into two sets: a finite set of “legitimate” strategies M'_i and a (possibly infinite) set of aborting strategies, all outcome-equivalent to each other so that they can be represented by a single strategy **abort**′. Then, strategy equivalence states that, for every player i , there is a one-to-one correspondence between $M_i \cup \{\mathbf{abort}\}$ and $M'_i \cup \{\mathbf{abort}'\}$, mapping **abort** to **abort**′, such that the outcome of any profile of strategies in \mathcal{M} is identical to the outcome of the profile of the corresponding strategies in \mathcal{M}' . Privacy equivalence states that no set of players, whether aborting or not, gains more information about the *non-aborting* strategies of the other players in \mathcal{M}' than they do in \mathcal{M} . (Aborting players publicly reveal that they have aborted, and no privacy protection applies to them.)

1.4 Answers to Selective Criticism

Privacy. Privacy has different meanings in different settings —economic settings of course, but also legal ones, psychological ones, and so on. Accordingly, some readers might be disappointed about the fact that no definitions of such versions of privacy can be found in our paper. Indeed, we have defined *privacy preservation*, rather than various “privacies.” We would like to stress, however, that our approach is a major

⁶Such a complete specification absorbs “aborts” into the abstract normal-form mechanism itself. But we are not constrained by this specific choice, and can adjust our construction to work with other specific ways to handle aborts.

advantage. Indeed, our definition of privacy preservation is so strong as to apply to economic privacy, legal privacy, psychological privacy, and indeed “privacy in any sense one might care about” —now or in the future. Adopting setting-sensitive definitions would have obliged us to provide different “perfect” mechanisms for the legal world, the psychological one, and so on. (And, quite possibly, we might have failed to deliver privacy-preserving mechanisms in some reasonable settings.)

Theory. Some readers may be concerned about the practicality of our result. Our paper is unapologetically theoretical. Indeed, we want to initiate a *rigorous* and *unified* treatment of privacy, trust, and efficiency in mechanism implementation. We focus on perfect implementations, admittedly the hardest case, as it is in the extreme that difficulties are best understood. Yet, our conceptual frameworks and techniques *a fortiori* apply to less-than-perfect implementation goals. But before negotiating compromises it is important to understand what our initial negotiating position can be. That is, it is important to understand what can be feasibly achieved with the maximum intransigence. This is indeed what our paper intends to do.

(We appreciate that a reason for future compromises is the inconvenience of randomizing ballots. It will indeed be useful to investigate what can be achieved with other primitives. But it should also be appreciated that there is a major distinction between our practical *inconvenience* and the practical *impossibility* of implementations such as *Naive* of Section 1.1.3.)

Proofs. Concerns might be raised about the length and non-traditional nature of our proofs. Let us point out, however, that solving *non-traditional problems* should be expected to require non-traditional concepts and techniques —else, we would already have *traditional solutions*. Besides, what is traditional depends on our current mind set. When concerns about privacy and efficiency will become integral part of our view of human interactions, our techniques and conceptual frameworks will appear quite natural. Also, while our proofs may be long in some “absolute” sense, it should be appreciated that they actually are very simple. Indeed, we just show how to perfectly implement each of 4 elementary functions: AND, NOT, COIN, and DUPLICATE. Because, every finite outcome function —no matter how complex— can be constructed from such building blocks, we actually provide a very short and natural proof that every normal-form mechanism can be perfectly implemented.

Simple Cases. Some readers may wonder why we do not *first* illustrate our construction for a simple case, such as the second-price mechanism. The reason is that, in order to guarantee the operational efficiency that we believe crucial for any meaningful theory of concrete implementation, we implement any outcome function g by implementing —separately, in the same order, and with a fixed sequence of elementary physical operations— each of the elementary computational operations involved in a computation of g . Thus, by the time we show how to perfectly implement each of AND, NOT, COIN, and DUPLICATE, we also show how to implement not only the second-price mechanism, but also *any mechanism whatsoever*. Hence, by illustrating our construction for some simpler mechanisms first, we would only duplicate our proofs.

Trust. Our construction could be misinterpreted as just replacing trust in the mediator with trust in Nature. More correctly, however, our construction replaces (vague) trust in *both* the mediator *and* Nature with (well-defined) trust in Nature alone. Indeed, even if the mediator were a former president (a rare and yet not universally trusted party), one would still need suitable physical assumptions to concretely implement normal-form mechanisms. Namely, in addition to the mediator’s trustworthiness, one would need to assume

- *The existence of a suitably secure form of “envelopes”* —as it is necessary for the players to privately transfer their reports to the mediator; and
- *The existence of a suitably secure form of a “randomization device”* —as it is necessary for the mediator to evaluate a probabilistic outcome function g in a way that at least keeps secret g ’s coin tosses.

Notice that, properly formalized, these are the very physical assumptions we rely on in our construction of perfect implementations. In a sense, therefore, we prove that by strengthening assumptions about *Nature*,

that are necessary anyway, we can dismiss all assumptions about *people*. That is, rather than capriciously and vaguely assuming trust on people or physical processes, *we distil the trust capable of guaranteeing perfect implementations*.

1.5 Analysis of Our Physical Assumptions

Since the benefits of our implementations (no trust, privacy, operational efficiency, etc.) derive from our physical assumptions, it is only proper to have a closer look at such assumptions. To begin with, it should be realized that physical assumptions are actually ubiquitous (at least implicitly) in most of game theory.

Physical Assumptions in Game Theory at Large. “Acts of Nature” (i.e., physical ways of sampling a given distribution outside of any possible influence of players or mediators) are implicitly necessary to the meaningfulness of very basic game theoretic concepts. For instance, games with public randomization rely on the existence of a *fair coin*. That is, such games *trust* Nature for the correctness (but not the privacy) of the elementary process consisting of the selection of a random bit. As clear and abstract fair coins can be, their existence is not guaranteed by Mathematics: it is a *physical assumption*.

More fundamentally for game theory, the meaningfulness of the very notion of a *mixed strategy* relies on a more complex act of Nature: the existence of a *private fair coin*. This time, Nature is trusted to provide each player i with a physical process capable of selecting a bit in a way that, not only is non influenceable by others, but is also *private* to i . Private coin tossing too is a physical assumption, and without it “secretly and randomly selecting an action” would be meaningless in practice. Indeed, if tossing a coin perturbed the universe in such a way that measurements made far away yielded information about the outcome of the coin toss, mixed strategies would be of no practical strategic significance.

Note that game theory often does not need to make its physical assumptions explicit. In fact, games with public randomization and mixed strategies are notions, and notions need not to be proved, but *motivated*. Quite differently, we not only wish to put forward the notion of a perfect implementation, but also prove that such implementations exist under some reasonable physical assumptions. Note too that a public coin is not adequate for our purposes, because the randomness used by the outcome function needs to remain secret. And a private fair coin is not appropriate either, since then we would need to trust the mediator to use the results of his privately obtained coin tosses.

Our Ballot Assumption. Our ballot assumption is formalized in Section 3. For now, just recall that we assume the existence of ballots of two different sizes enjoying two main properties: *security*, that is, each ballot perfectly hides and guarantees the integrity of its contents; and *indistinguishability*, that is, ballots of the same size are identical from the outside. (We actually need only the ballots used by the verifiable mediator to be indistinguishable.) The difference in size enables us to nest up to 5 smaller ballots into a larger one. We actually refer to smaller ballots as envelopes and to larger ones as super-envelopes, and thus use the term “ballot” when we do not want to distinguish between envelopes and super-envelopes.

Identical assumptions about ballots have been made by Ben-Porath (1998), and more explicitly and with additional explanations by Krishna (2006) —cfr. pages 2, 6 and 7.

(Envelopes with just the security property are absolutely common in pre-play communication literature.⁷)

Our Ballot-Box Assumption. Our ballot-box assumption too is formalized in Section 3. Recall that, intuitively, it states that it is possible to randomize a sequence of identical envelopes so that no one —player or mediator— can control or predict the permutation used in the process. In practice, such act of Nature might be realized by using spherical ballots, inserting them in a bingo-like machine, and cranking it several times. But: *Is it possible to achieve our goals without this —or an equivalent— physical assumption?*

⁷The papers that do not explicitly rely on envelopes, do rely on abstract channels of communication, assumed to guarantee (1) the integrity of the message, (2) the secrecy of the message, (3) the identity of the sender, and (4) the “auditability” of past messages. All properties that are indeed guaranteed by envelopes.

We conjecture that the answer is *NO*, and wish to motivate our answer by analyzing an elementary case. Let G be the following two-player game, and E be the correlated equilibrium of G which assigns probability $1/5$ to each of outcomes (A, A) , (B, B) , (C, C) , (C, D) and (D, C) . Consider now the problem of concretely implementing E .

	A	B	C	D
A	9, 6	-100, -100	-100, -100	-100, -100
B	-100, -100	6, 9	-100, -100	-100, -100
C	-100, -100	-100, -100	4, 4	1, 5
D	-100, -100	-100, -100	5, 1	0, 0

Providing such concrete implementations was the original motivation of Ben-Porath’s and Krishna’s for assuming perfect (two-size) ballots. Their solutions were very inefficient, but this does not transpire in the simple case at hand. Indeed, to implement E , their protocol requires constructing 5 super-envelopes, each containing two distinct envelopes, one marked “player 1” and the other marked “player 2”.⁸ The envelopes of the first super-envelope respectively contain “A” and “A”, those of the second super-envelope contain “B” and “B”, those of the third “C” and “C”, those of the fourth “C” and “D”, and those of the fifth “D” and “C”. Following Ben-Porath, Krishna recommends that one of the players secretly prepares all such envelopes and super-envelopes, and that afterwards a “joint lottery” is used to (1) with suitably high probability, inspect all envelopes and super-envelopes, and verify that their contents are as prescribed, and (2) with complementary probability, select at random a super-envelope e from the 5 possible ones. But since all envelopes, super-envelopes, and their contents can be publicly prepared, for simplicity it suffices to consider a joint lottery whose only goal is to pick a super-envelope e at random. As it is intuitively clear, if such e is then opened and its first (still sealed) envelope is given to player 1 and its second envelope to player 2, then both players, by privately opening their own envelopes, enter in possession of recommendations correlated according to E , and have no incentive to deviate from them in the immediately following play of G . However: *how should e be selected?* We consider two possible answers:

- The simplest option (adopted by us) is to rely on our ballot-box assumption. Essentially: have a bingo-like device return the 5 super-envelopes in random order, and then pick the first one.
- Another option (adopted by Krishna) is to rely on an elementary pre-play protocol. Essentially: first player 1, in private, randomly permutes the 5 super-envelopes; then player 2, in private, randomly picks one of them.

Thus the first option requires a physical assumption beyond the existence of perfect ballots, while the second one does not appear to do so. In principle, of course, the fewer the assumptions the better. But two main reasons make the first option preferable, if one wishes to *provably* construct a strong implementation of E .

1. *Hidden Assumptions.* Krishna does not formalize the exact physical assumptions behind his lottery mechanism. But specifying such assumptions would be necessary if one wanted to *formally prove* that the lottery works. In particular, after being handed the 5 super-envelopes already (suppositively) randomized by player 1, player 2 must be given some privacy to randomly pick one of them. Else, player 1 will know which e is picked and thus player 2’s recommendation. Thus, once the envelopes are no longer in public view, the danger exists that player 2 may open and reseal the super-envelopes and then choose the one he prefers (or even “emerge” with a completely different super-envelope that he prepared in advance, or...). Accordingly, a rigorously analyzable lottery protocol would ultimately require additional physical assumptions of its own: for instance, a *perfect sealing* property, that is, an assumption guaranteeing that any physical way of reading e ’s content will leave a public and non-erasable trace.

⁸Rather than marking envelopes, in our paper we let super-envelopes have a “top” side and be only slightly larger than the envelopes they contain. Thus, upon by opening them one recovers their (sub)envelopes in the same order in which they were originally inserted.

2. *Failure of Strong Implementation.* Krishna’s objectives are essentially limited to “expanding the set of equilibrium outcomes via cheap-talk.” It is thus not surprising that his lottery mechanism allows rational players to reach, *without any outside communication*, outcomes not distributed according to E and *not reachable otherwise*. But this clashes with our goals. In mechanism design, the traditional objective, and the designer’s duty, is to “make it in the players’ interest to obtain the outcomes desired by Society.” Assuming now that Society’s interest is to have G ’s outcomes distributed according to E , let us illustrate how Krishna’s mechanism fails to reach this objective in a strong sense: that is,

It fails to match the set of equilibrium outcomes reachable in G with the assistance of a private mediator generating a profile of private recommendations distributed according to E (and E only).

Indeed, the payoffs being as described, the players would much prefer the correlated equilibrium E' which assigns probability $1/2$ to each of outcomes (A, A) and (B, B) . Note that E' actually is the *best* correlated equilibrium from the players’ combined perspectives, and that it cannot be achieved “with E -recommendations alone.” Now, the problem is that the above lottery protocol makes it trivial for the two players to implement E' (instead of E) without Society realizing what is going on. This can be done as follows. Rather than randomly permuting all 5 super-envelopes, player 1 first randomizes the two envelopes respectively containing the recommendations (A, A) and (B, B) , and puts them in positions 1 and 2. Then he randomizes the other 3 super-envelopes and puts them in positions 3, 4, and 5. At this point, player 2 randomly selects one of the first two super-envelopes. Accordingly, (a) by so acting the players implement E' rather than E ; (b) neither player has an incentive to deviate; and (c) Society cannot observe that something is about to go wrong. Occasional inspection of all envelopes only verifies that the proper contents were there, but not whether their positions looked “suspicious.” (Besides, can one reject as “non random” a permutation of 5 elements whose first two elements “remain local?”)

Of course, by way of hindsight, to prevent the players from hijacking Society’s desired equilibrium into the direction they want, one might envisage adding an outside entity that further randomizes the super-envelopes immediately after player 1. However, this additional randomization cannot happen in slow motion and full public view, as the players would be able trace the positions of the super-envelopes. And if such outside entity is envisaged to *privately* perform this randomization, then he/she/it must be *assumed* to do so correctly. Ultimately, therefore, such an external entity should be trusted at least as much as our ballot box!

Could the ballot box become redundant if there were more players? That is: could a formal assumption weaker than the ballot box provably guarantee *strong* implementation of correlated equilibrium in games with three or more players? Very hard to say. Strong implementations (never mind perfect ones!) ultimately require analyzing the strategic opportunities of *groups* of players. And the higher the number of players, the harder this analysis gets. With more players, deviations from the envisaged strategies much more subtle than those discussed above may lead to more unexpected consequences.

Completeness of Our Assumptions. It should be realized that correlated equilibrium does not require any private inputs from the players. And if unexpected consequences arise in concrete implementations of correlated equilibrium, one can only speculate what might go wrong in more general incomplete-information settings and with much more stringent goals. As we just saw, our ballot-box assumption was naturally brought in to resolve a specific problem with the above joint lottery. If fixing each glitch in concrete implementations required a new physical assumption, one would risk ending up with an infinite list of natural requirements, since there may not be an upper-bound to the number of possible glitches. Therefore, the fact that perfect ballots and a ballot box suffice to construct perfect implementations for any possible normal-form mechanism is somewhat surprising. In a sense, our result proves the “completeness” of a very small set of physical axioms.

Reasonableness of Our Assumptions. The simplicity, completeness and long history of ballots and ballot boxes may be appealing, but ultimately: Are these assumptions true? Of course this is impossible to know in a mathematical sense, and perhaps no true envelopes, ballot boxes, or fair coins exist in Nature. Thus, a more pertinent question is: *Are our physical axioms reasonable?*

Certainly, identical ballots and their randomizers are reasonable enough to have been relied upon for a long, long time. Not only are they the very ingredients for fair lotteries and tallying secret votes, but also for *card playing*. Indeed, similarly to our ballots, cards can be viewed as a sequence of objects perfectly equal on one side and carrying distinct, secret and unalterable contents on the other side. And card playing relies on an act of Nature for randomization. Is there such an act of Nature? Perhaps not, but it is certain that, for centuries, people and casinos have been behaving as if there is. Thus it may be worth to re-state our result in the following alternative way:

“Any normal-form mechanism has a perfect implementation if and only if playing cards is possible.”

1.6 Road Map

After discussing the connection of our work with existing literature in Section 2, we proceed to prove that perfect ballot-box implementations exist in stages. In Section 3 we formalize ballots and ballot-boxes, and in Section 4 the notion of a public mediator. Then, in Sections 5 through 7 we present our main technical result. Namely that it is possible for a public mediator, given a sequence of envelopes whose contents properly encode inputs x_1, \dots, x_n , to publicly manipulate these envelopes so as to produce a new sequence of envelopes that are guaranteed to contain the proper encoding of $g(x_1, \dots, x_n)$ without revealing no information whatsoever about the inputs. Then, in Section 8 we present how a public mediator, again without revealing anything, can check whether a sequence of “input envelopes” contains a properly encoded input. Finally, in Section 9 we formalize the notion of a perfect implementation and prove that (1) having the players submit input envelopes encoding their private reports, (2) having the mediator check that the input envelopes contain proper encodings, (3) having the mediator transform input envelopes into output envelopes encoding g ’ output, and finally opening them, constitutes a perfect implementation of the finite normal-form mechanism with public outcomes whose outcome function is g . (In Appendix C we show how to extend our construction to normal-form mechanisms that also have private outcomes.) Section 10 concludes.

2 Relation to Prior Work

2.1 Zero Knowledge and Secure Computation

The study of privacy and correctness without trust started two decades ago, in theoretical computer science, with the zero-knowledge proofs of Goldwasser, Micali and Rackoff (1985). Assume that a prover P wishes to prove a mathematical statement S to a verifier V , keeping 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”.⁹ 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.

Another body of work on privacy and correctness without trust is *secure computation*, as introduced by Goldreich, Micali and Wigderson (1987) (improving on earlier results of Yao (1986)). In essence, they show that, for any finite function f with n inputs, there exists a communication protocol P_f such that, whenever the majority of n players stick to P_f ’s instructions, f can be evaluated by the players alone so as to match the privacy and correctness provided by the following process involving a private mediator M :

1. After having the opportunity to secretly communicate with the others, each player i privately hands to M either an aborting signal or a private input x_i .
2. If M receives an aborting signal, then he announces “the evaluation of f has been aborted.” Else, he announces the output y obtained by privately evaluating f on the n received inputs.

⁹For 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)!

Letting f be the outcome function of a normal-form mechanism \mathcal{M} , at least two main problems prevent a secure computation of f from being a perfect implementation of \mathcal{M} .

The first problem is *strategic signalling*. Secure computation suffers from this problem because, as summarized above, it envisages that the players secretly communicate to each another beforehand. Of course, one could remove *by fiat* any secret inter-player communication in the above mediated process for evaluating f , but then no known secure-computation protocol P_f would match such a modification.¹⁰ As we have discussed in Section 1.3, perfect implementations are not affected by strategic signalling.

The second problem is *collusive power*. Although coalitions of less than half of the players are powerless, coalitions of more than half of the players are “omnipotent:” that is, without any recourse, they can force any output of f they want, no matter what inputs the non-collusive players might have chosen. In essence, in any secure computation protocol P_f , any majority of the players enjoy the same power that players 1 and 2 have in *Extreme SPA* of Subsection 1.1.4.¹¹

(Accordingly, our results can be interpreted as providing a stronger notion of secure computation, namely one that is that is immune to strategic signalling and collusion neutral.)and

2.2 Games with Pre-Play Communication

Our result can be viewed as generalizing and strengthening the extensive body of work on pre-play communication games —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 (2007), Krishna (2006). Such games have been designed for achieving correlated equilibrium, communication equilibrium, or Bayesian-Nash equilibrium.

The general objective of these papers is obtaining (1) the equilibrium payoffs of a given normal-form game G played with the help of a private mediator, by means of (2) the equilibrium payoffs of G extended by a carefully designed communication protocol. To explain why a perfect implementation of the same mediated version of G guarantees much more, let us summarize the pre-play approach in a way that enables an easy comparison with ours. We start by briefly explaining the mechanics of their mediated and extended games.

Let G be a normal-form game for n players, and f a probabilistic function mapping n inputs to a strategy profile (s_1, \dots, s_n) of G . (As in the case of correlated equilibrium, f 's inputs may be empty.) Then G can be extended in the following two ways.

- By (f, G) we denote the privately mediated game played in three stages as follows. In the first stage, the players simultaneously provide their f -inputs to a private mediator. In the second stage, the mediator obtains a profile of strategies (s_1, \dots, s_n) by privately evaluating f on the received inputs, and then

¹⁰Indeed, secure-computation protocols have been developed in two main communication models: broadcasting and private channels. In the latter model, developed by Ben-Or, Goldwasser and Wigderson (1988) and Chaum, Crépeau and Damgård (1988), it is envisaged that between each pair of players i and j there exists a dedicated “lead pipe” that enables 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, nor observe whether they are communicating at all. A secure-computation protocol P_f in this model therefore enables secret inter-player communication in the easiest possible way. In the broadcasting model —indeed the communication model originally envisaged by Goldreich et al. (1987)— players speak in turns, and the identity of each speaker and what he says becomes common knowledge. (In this setting privacy relies on encryption, and thus security is computational rather than information theoretic.) Yet, secret inter-player communication is still inherent. Indeed, all secure-computation protocols require the players to execute probabilistic strategies, and thus exchange messages with positive “entropy.” Whenever this is the case, 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 an evaluation of g with a trusted mediator. (Although this is not in the scope of the present paper, we notice that we can modify the protocol of Goldreich et al. (1987) so as to make signalling inconsequential for the current evaluation of g .)

¹¹To be sure, Goldreich, Micali and Wigderson also put forward a weaker notion of secure computation where no subset of players can control f 's output. But then any single player can prevent f 's correct output to become public knowledge, for any non-trivial f .

privately hands s_i to player i as his “recommendation.” In the third stage, the players play G .¹²

- By (P, G) we denote the game played as follows. First, the players communicate following a communication protocol P_f , so that each player i computes a strategy s_i in G . Then, they play G . (The protocol P requires specific communication channels, such as exchanging messages in sealed envelopes.)

The function f is specifically chosen so that (f, G) has a desired equilibrium, $E_{f,G}$, in which the players *play their recommended strategies*. With this focus, therefore, the function f itself becomes a normal-form mechanism. Indeed, after the players simultaneously select an input profile (x_1, \dots, x_n) for f , the outcome consists of the strategy profile $(s_1, \dots, s_n) = f(x_1, \dots, x_n)$ in G .

With this preamble, the goal of pre-play is to come up with a communication protocol P such that, sticking to P 's instructions and then playing the computed strategies in G is an equilibrium $E_{P,G}$ of (P, G) , having the same payoffs as $E_{f,G}$.¹³ Thus, pre-play may be viewed as providing a concrete implementation of the normal-form mechanism f .

Let us now highlight three main differences between pre-play and perfect implementations, starting with the two easier to describe.

- **Operational Inefficiency.** In all pre-play literature, protocol P_f treats f as a table, and thus requires an amount of physical resources that is exponential in the complexity of f . By contrast, a perfect implementation of f requires a number of physical operations that is linear in f 's complexity.
- **Restrictions on the Players.** To achieve correlated and communication equilibria that are not convex combinations of, respectively, Nash and Bayesian-Nash equilibria, the constructions of Bárány (1992), Forges (1990), Ben-Porath (1998), Aumann and Hart (2003), Ben-Porath (2003), Gerardi (2004), Gerardi and Myerson (2007) require at least 3 players. The other cited pre-play constructions work for 2 players, but restrict the rationality of the players by assuming that they cannot solve computationally complex problems—such as breaking codes.¹⁴ No such restrictions are required for perfect implementations.
- **Strategy Inequivalence.** The requirement that (P, G) has equilibria that are payoff-equivalent to some or even all the equilibria of (f, G) is much weaker than strategy equivalence. In fact, (P, G) may *introduce* additional equilibria, that is, possess equilibria having no counterparts in (f, G) . Therefore, a rational play of (f, G) and a rational play of (P, G) may be vastly different.

Again, one main way for (P, G) to introduce additional equilibria is via signaling. Indeed, while in the mediated game (f, G) the players do not interact with each other, in (P, G) they must do so, and have plenty of opportunity to signal to each other. (In most pre-play constructions, the players are actually envisaged to communicate via private channels.)

But even if f and G were such that signaling could not introduce additional equilibria in (P, G) , playing (f, G) may be vastly different from playing (P, G) . This may be the case *even if*, in the extreme, for each equilibrium of (f, G) there exists a payoff-equivalent equilibrium of (P, G) , *and viceversa!* The reason is that, even in such an extreme case, “equilibrium selection” may still be very different in the two games. We illustrate this seemingly counterintuitive phenomenon relative to achieving correlated equilibrium via pre-play. Consider the following normal-form game.

G : There are 4 players, each having action set $\{a, b\}$.

The payoffs of strategy profile (a, a, a, a) is $(0, 1, 0, 1)$;

¹²One might consider more general versions of (M_f, G) , where the players and the mediator M_f interact back and forth. Forges (1986) and Myerson (1982) show that it suffices to consider *canonical* mechanisms, where the players communicate with M_f in three stages as described.

¹³The proof techniques often deliver more. In particular, at equilibria $E_{f,G}$ and $E_{P,G}$, the distributions of—respectively—the recommended and computed strategies are the same.

¹⁴Other types of restrictions are also present. For instance, Dodis et al. (2000) relies on min-max punishments and thus on empty threats. Without loss of generality, assume that player 1 is the first, in the pre-play, to privately realize what his recommendation is. If he does not like his recommendation (i.e., if the expected payoff conditioned on his recommendation is bad for him), he may abort all communication, thus preventing player 2 from learning his own recommendation. Min-max punishment is certainly a way to prevent this, but may not be rational once player 1 aborts.

that of (b, b, b, b) is $(1, 0, 1, 0)$;
that of (a, b, a, b) is $(10, 10, -100, -100)$; and
that of any other strategy profile is $(-\infty, -\infty, -\infty, -\infty)$.

Note that G has the following “attractive” correlated equilibrium: $E = \frac{1}{2}(a, a, a, a) + \frac{1}{2}(b, b, b, b)$, which in expectation gives each player a payoff of $\frac{1}{2}$. Now,

- Let f the function that, on no inputs, returns a 4-tuple of strategies distributed as in E ; and
- Let P be the following *naive* protocol: First players 1 and 2 flip a fair coin in a private room to obtain a private random bit c . Then player 1 privately sends c to player 3, and player 2 privately sends c to player 4. All players interpret $c = 0$ as the recommendation a , and $c = 1$ as the recommendation b .

Then it is immediately seen that

- For any equilibrium e of (f, G) there exists a payoff-equivalent equilibrium e' in (P, G) , and viceversa.
- By colluding in (P, G) —and in a private room it is easy to collude!— players 1 and 2 can “hijack any equilibrium E' so that (a, b, a, b) is played.”

Indeed, despite the fact that the payoffs of (a, b, a, b) are extremely unattractive to players 3 and 4, it suffices for player 1 to play a and induce player 3 to do the same by sending him 0, and for player 2 to play b and induce player 4 to do the same by sending him 1.

- In (f, G) , even if players 1 and 2 were given for free means of private communication, it would be impossible for them to collude so as to “hijack E into (a, b, a, b) .”

Accordingly, although in (f, G) the equilibrium E is likely to be selected, in (P, G) —because it is so easy and profitable to hijack E' into (a, b, a, b) — E' is unlikely to be played out. The example illustrates that preserving all equilibrium payoffs can be too weak a goal for concrete implementations in some settings. Only strategy equivalence guarantees that the players are always indifferent between playing a mediated mechanism or its concrete implementation.

Of course, although the above P satisfies all the stated objectives of a pre-play implementation of the above f , one might suspect that the above equilibrium-selection problem arises because P is unnecessarily naive. In other words, one may suspect that the actual pre-play techniques “deliver more than claimed.” Unfortunately, this is not the case. The same problem would arise if P were chosen to be any pre-play implementation of f in the cited literature. Although much more general (i.e., capable of handling functions with true private inputs), sophisticated, and sequentially rational, any such implementation of f enables a subset of the players to control the output of f without being even suspected by the other players. (In essence, such players have a collusive power similar to that enjoyed by players 9 and 10 in the concrete mechanism *Extreme SPA* of Subsection 1.3).

Secure Computation vs. Pre-Play. The above discussed weaknesses (relative to perfect implementation!) of pre-play implementations are similar to those discussed for secure computation. This is not by chance. Retrospectively, in fact, one can prove that secure computation suffices to guarantee pre-play’s goals. Namely, if the number of players, n , is greater or equal to 3, P_f is a secure-computation protocol for a n -input function f , then, for any equilibrium E of (f, G) , (P_f, G) has an equilibrium that is payoff-equivalent to E . In fact, a weaker notion of secure computation suffices to guarantee this implication.¹⁵

¹⁵As recalled in Section 2.1, in any secure-computation implementation of f , correctness and privacy are guaranteed no matter how a coordinated minority of the players deviates from their communication instructions. To guarantee pre-play’s goals it actually suffices to consider secure computations that withstand coalitions of cardinality 1, because equilibrium conditions solely check for individual deviations.

2.3 Other Prior Work

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*.

3 The Ballot-Box Model

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 reasonable actions available to the players. In this paper, we make a specific choice, sufficient for our present goals.

3.1 Intuition

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 ballot-box actions. Each action in the first 7 classes is referred to as a *public action*, because it is performed 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; (6) publicly destroying a ballot; and (7) do nothing. The last three 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 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 Information. Conceptually, the players observe which actions have been performed on which ballots. Formally, (1) we associate 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 —e.g., after being ballot-boxed); and (2) we have each action generate, when executed, a public string of the form “ A, j, k, l, \dots ”; where A is a string identifying the action and j, k, l, \dots are the identifiers of the ballots involved. The *public record* is the concatenation of the public strings generated by all actions executed thus far.

3.2 Formalization

Basic Notation. We denote by Σ the alphabet consisting of English letters, arabic numerals, and punctuation marks; by Σ^* the set of all finite strings over 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 $p := \text{rand}(\mathbb{S}_k)$ the operation that assigns to variable p a randomly selected permutation in \mathbb{S}_k ; and by \emptyset the empty set.

If S is a set, by S^0 we denote the empty set, and by S^k the Cartesian product of S with itself k times. If x is a sequence, by either x^i or x_i we denote x 's i th element,¹⁶ and by $\{x\}$ the set $\{z : x^i = z \text{ for some } i\}$. 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 i th 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 xy .

If A is a probabilistic algorithm, the distribution over A 's outputs on input x is denoted by $A(x)$. A probabilistic function $f : X \rightarrow Y$ 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.

Ballots and Actions. 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.)

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*).

Relative to a well-defined set of ballots B : if j is an envelope in B , then $\text{cont}_B(j)$ denotes the content of j ; if $x = j^1, \dots, j^k$ is a sequence of envelope identifiers in I_B , then $\text{cont}_B(x)$ denotes the concatenation of the contents of these envelopes, that is, the string $\text{cont}_B(j^1) \cdots \text{cont}_B(j^k)$.

A *global memory* consists of a pair (B, R) , where

- B is a well defined set of ballots; and
- R is a sequence of strings in Σ^* , $R = R^1, R^2, \dots$

We refer to B as the *ballot set*; to R as the *public record*; and to each element of R as a *record*. The *empty global memory* is the global memory for which the ballot set and the public record are empty. We denote the set of all possible global memories by GM .

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

¹⁶For any given sequence, we shall solely use superscripts, or solely subscripts, to denote all of its elements.

constructions); and a functional specification. If $gm = (B, R)$, we actually specify $a(gm)$ as a program acting on variables B and R . For convenience, we include in R the auxiliary variable \mathbf{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 .

1. (NEWEN, c) —where $c \in \Sigma$.
 “Make a new envelope with public content c .”
 $\mathbf{ub} := \mathbf{ub} + 1$; $B := B \cup \{(\mathbf{ub}, c, 0)\}$; and $R := R \circ (\text{NEWEN}, c, \mathbf{ub})$.
2. (OPENEN, j) —where j is an envelope identifier in I_B .
 “Publicly open envelope j to reveal content $\text{cont}_B(j)$.”
 $B := B \setminus \{B_j\}$ and $R := R \circ (\text{OPENEN}, j, \text{cont}_B(j), \mathbf{ub})$.
3. (NEWSUP, j_1, \dots, j_L) —where $L \leq 5$, and $j_1, \dots, j_L \in I_B$ are distinct envelope identifiers.
 “Make a new super-envelope containing the envelopes j_1, \dots, j_L .”
 $\mathbf{ub} := \mathbf{ub} + 1$; $B := B \cup \{(\mathbf{ub}, (\text{cont}_B(j_1), \dots, \text{cont}_B(j_L)), L)\}$;
 $B := B \setminus \{B_{j_1}, \dots, B_{j_L}\}$; and $R := R \circ (\text{NEWSUP}, j_1, \dots, j_L, \mathbf{ub})$.
4. (OPENSUP, j) —where $j \in I_B$ is the identifier of a super-envelope of level L .¹⁷
 “Open super-envelope j .”
 letting $\text{cont}_B(j) = (c_1, \dots, c_L)$, $B := B \cup \{(\mathbf{ub} + 1, c_1, 0), \dots, (\mathbf{ub} + L, c_L, 0)\}$; $B := B \setminus \{B_j\}$;
 $\mathbf{ub} := \mathbf{ub} + L$; and $R := R \circ (\text{OPENSUP}, j, \mathbf{ub})$.
5. (PUBLICPERMUTE, j_1, \dots, j_k, p) —where $k \leq 5$, $p \in \mathbb{S}_k$, and $j_1, \dots, j_k \in I_B$ are distinct identifiers of ballots with the same level L .
 “Publicly permute j_1, \dots, j_k according to p .”
 $B := B \cup \{(\mathbf{ub} + 1, \text{cont}_B(j_{p(1)}), L), \dots, (\mathbf{ub} + k, \text{cont}_B(j_{p(k)}), L)\}$; $B := B \setminus \{B_{j_1}, \dots, B_{j_k}\}$;
 $\mathbf{ub} := \mathbf{ub} + k$; and $R := R \circ (\text{PUBLICPERMUTE}, j_1, \dots, j_k, p, \mathbf{ub})$.
6. (DESTROY, j) —where j is a ballot identifier in I_B .
 “Destroy ballot j ”
 $B := B \setminus \{B_j\}$ and $R := R \circ (\text{DESTROY}, j, \mathbf{ub})$.
7. (DONOTHING).
 “Do nothing”
 $B := B$ and $R := R \circ (\text{DONOTHING}, \mathbf{ub})$.
8. (BALLOTBOX, j_1, \dots, j_k) —where $k \leq 5$ and $j_1, \dots, j_k \in I_B$ are distinct identifiers of ballots with the same level L .
 “Ballotbox j_1, \dots, j_k ”
 $p := \text{rand}(\mathbb{S}_k)$; $B := B \cup \{(\mathbf{ub} + p(1), \text{cont}_B(j_1), L), \dots, (\mathbf{ub} + p(k), \text{cont}_B(j_k), L)\}$;
 $B := B \setminus \{B_{j_1}, \dots, B_{j_k}\}$; $\mathbf{ub} := \mathbf{ub} + k$; and $R := R \circ (\text{BALLOTBOX}, j_1, \dots, j_k, \mathbf{ub})$.

We refer to a member of the first 7 classes as a *public action* and to a member of the 8th class as an *action of Nature*.

¹⁷All 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.

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 1. A global memory gm is feasible if there exists a sequence of global memories gm^0, gm^1, \dots, 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) is a feasible memory, we refer to R as a feasible public record.

Notice that if $gm = (B, R)$ 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 2. 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 .

4 The Notion of a Public Ballot-Box Mediator

Definition 3. Let \mathcal{P} be a sequence of K functions. We say that \mathcal{P} is a public ballot-box mediator (of length K) if, for all $k \in [1, K]$ and public records R , $P^k(R)$ is a public ballot-box action in \mathcal{A}_R .

An execution of \mathcal{P} on an initial feasible global memory (B^0, R^0) is a sequence of global memories $(B^0, R^0), \dots, (B^K, R^K)$ such that $(B^k, R^k) = a^k(B^{k-1}, R^{k-1})$ for all $k \in [1, K]$, where $a^k = P^k(R^{k-1})$.¹⁸

If e is an execution of \mathcal{P} , by $B^k(e)$ and $R^k(e)$ we denote, respectively, the ballot set, the public record, and the private history profile of e at round k . By $R_{\mathcal{P}}^k(e)$ we denote the last k records of $R^k(e)$ (i.e., “the records appended to R^0 by executing \mathcal{P} ”).

Remarks.

- Note that the above definition captures our intuitive desideratum that no special trust is bestowed on a public mediator. Because he performs a sequence of public ballot-box actions, any one can verify that
 - (i) he performs the right sequence of actions;
 - (ii) he does not choose these actions; and
 - (iii) he does not learn any information that is not publicly available.
- Note too that if $\mathcal{P} = P^1, \dots, P^K$ and $\mathcal{Q} = Q^1, \dots, Q^L$ are public mediators, then their concatenation, that is, $P^1, \dots, P^K, Q^1, \dots, Q^L$ is a public mediator too.

5 The Notion of a Ballot-Box Computer for a Function f

Definition 4. An address is a finite sequence x of distinct positive integers. An address vector x is a vector of mutually disjoint addresses, that is, $\{x_i\} \cap \{x_j\} = \emptyset$ whenever $i \neq j$. The identifier set of an address vector $x = (x_1, \dots, x_k)$ 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

¹⁸Note that the executions of \mathcal{P} are, in general, random since $P^k(R)$ may return an action of Nature.

$cont_B(x)$ to be the vector $(cont_B(x_1), \dots, cont_B(x_k))$. If i is a positive integer, then $x+i$ is the address vector whose j th 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.

A ballot-box computer \mathcal{P} for a function f is a special public ballot-box mediator. 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 the initial memory is not of the proper form.

Definition 5. Let $f : X^a \rightarrow Y^b$ be a finite function, where $X, Y \subset \Sigma^*$; and let $x = x_1, \dots, x_a$ be an address vector. We say that a feasible global memory $gm = (B, R)$ is proper for f and x if $I_x \subset I_B$ and $cont_B(x) \in X^a$.

Definition 6. Let $f : X^a \rightarrow Y^b$ be a finite function, where $X, Y \subset \Sigma^*$; let x and w be two address vectors. We say that a public ballot-box mediator \mathcal{P} is a ballot-box computer for f , with input address vector x and output address vector w , if there exists a probabilistic algorithm SIM such that, for any execution e of \mathcal{P} on an initial memory $gm^0 = (B^0, R^0)$, proper for f and x and with identifier upper-bound ub_0 , the following three properties hold:

1. Correctness: $cont_{B^K(e)}(w + \mathbf{ub}) = f(cont_{B^0}(x))$.
2. Privacy: $R_{\mathcal{P}}^K(e) = SIM(\mathbf{ub})$.
3. Clean Operation: $B^K(e) = B_{\{w+\mathbf{ub}\}} \cup B^0 \setminus B_{\{x\}}$.

We refer to SIM as \mathcal{P} 's simulator; to $B_{\{x\}}$ as the input envelopes; and to $B_{\{y+\mathbf{ub}\}}$ 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, Correctness implies that each integer of each address $w_j + \mathbf{ub}$ 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 $R_{\mathcal{P}}^K$, 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 \mathbf{ub} is deducible from R^0 , and thus already known to the players, and (2) $R_{\mathcal{P}}^K$ can be generated “with exactly the same odds as in a random execution of \mathcal{P} ” by \mathcal{P} 's simulator only on input \mathbf{ub} . Since such simulator is a *fixed algorithm*, $R_{\mathcal{P}}^K$ cannot contain any information about f 's inputs and outputs that is not deducible from \mathbf{ub} .
- *Clean Operation.* As we shall see, our construction is modular, that is, we build computers for complex functions from computers for more elementary ones. Therefore, 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 “touch” any ballots of the initial memory besides its input envelopes. This way, partial results already computed, if any, will remain intact. Accordingly, 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).

Clean Operation 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.¹⁹

6 Three Elementary Ballot-Box Computers

In this section we first provide ballot-box computers for three elementary functions. (These computers will later on be used as building blocks for constructing computers for arbitrary finite functions.) Our three elementary functions are:

1. *Permutation Inverse*, mapping a permutation $p \in \mathbb{S}_5$ to p^{-1} .
2. *Permutation Product*, mapping a pair of permutations $(p, q) \in \mathbb{S}_5 \times \mathbb{S}_5$ to pq —i.e., the permutation of \mathbb{S}_5 so defined: $pq(i) = p(q(i))$.
3. *Permutation Clone*, mapping a permutation $p \in \mathbb{S}_5$ to the pair of permutations (p, p) .

ENCODINGS. Note that the notion of a ballot-box computer \mathcal{P} for f applies to functions f from strings to strings. (Indeed, f ’s inputs and outputs must be represented as the concatenation of, respectively, the symbols contained in \mathcal{P} ’s input and output envelopes.) Thus we need to encode the inputs and outputs of Permutation Inverse, Product and Clone as strings of symbols. This is naturally done as follows.

Definition 7. We identify a permutation p in \mathbb{S}_5 with the 5-long string $p_1p_5PA_3p_4p_5$, such that $p_j = p(j)$. Relative to a well-defined set of ballots B , we say that a sequence σ of 5 envelope identifiers is an envelope encoding of a permutation if $\text{cont}_B(\sigma) \in \mathbb{S}_5$.

If σ is an envelope encoding of a permutation in \mathbb{S}_5 , we refer to this permutation by $\hat{\sigma}$. We consistently use lower-case Greek letters to denote envelope encodings.

PUBLIC-MEDIATOR CONVENTIONS. To simplify our description of a public mediator \mathcal{P} we adopt the following conventions.

- Rather than describing \mathcal{P} as a sequence of K functions that, on input a public record R , output a ballot-box action feasible for any global memory with public record R , we present \mathcal{P} as a list of K actions a^1, \dots, a^K (to be performed no matter what the public record may be). Should any such a^k be infeasible for a particular global memory, we interpret it as the “do nothing” action, which is always feasible.
- We describe each action a^k via its informal reference (as per Definition 3.2), using an explicit and convenient reference to the identifiers it generates. For instance, when we say “Make a new envelope x with public content c ”, we mean (1) “Make a new envelope with public content c ” and (2) “refer to the identifier of the newly created envelope as x ” —rather than $\text{ub} + 1$.
- 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 .

¹⁹For 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}' player i can ex-post prove how he voted to some other player j by handing over 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.

6.1 A Ballot-Box Computer for Permutation Inverse

Public mediator $\mathcal{IN}\mathcal{V}_\sigma$

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, \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 ν_ℓ , and denote its content by $\hat{\nu}_\ell$. Set $\hat{\nu} = \hat{\nu}_1 \circ \dots \circ \hat{\nu}_5$.
- (6) Publicly permute μ_1, \dots, μ_5 according to $\hat{\nu}^{-1}$ to obtain ρ_1, \dots, ρ_5 . Set $\rho = \rho_1, \dots, \rho_5$.

Output address: 26, 27, 28, 29, 30.

Lemma 1. *For any 5-long address σ , $\mathcal{IN}\mathcal{V}_\sigma$ is a ballot-box computer for permutation inverse, with input address σ and output address 26, \dots , 30.*

Proof. As per Definition 6, let us establish Correctness, Privacy and Clean Operation for $\mathcal{IN}\mathcal{V}_\sigma$. Consider an execution of $\mathcal{IN}\mathcal{V}_\sigma$ on any initial memory gm^0 proper for permutation inverse and σ , and let ub^0 be the identifier upper-bound of gm^0 .

Correctness. Step 1 generates 5 new identifiers (increasing ub^0 by 5). Step 2 binds together, in the same super-envelope A_ℓ , the ℓ th 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 $\mu = \mu_1, \dots, \mu_5$ and $\nu = \nu_1, \dots, \nu_5$ 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, $\rho = \text{ub}^0 + 26, \dots, \text{ub}^0 + 30$; 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 $SIM-\mathcal{IN}\mathcal{V}_\sigma$

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 $\mathcal{IN}\mathcal{V}_\sigma$):

- (NEWEN, 1, $ub + 1$) \dots (NEWEN, 5, $ub + 5$)
 (NEWSUP, $ub + 1, \sigma_1, ub + 6$) \dots (NEWSUP, $ub + 5, \sigma_5, ub + 10$)
 (BALLOTBOX, $ub + 6, \dots, ub + 10, ub + 15$)
 (OPENSUP, $ub + 11, ub + 17$) (OPENSUP, $ub + 12, ub + 19$) \dots (OPENSUP, $ub + 15, ub + 25$)
 (OPENEN, $ub + 17, r_1, ub + 25$) (OPENEN, $ub + 19, r_2, ub + 25$) \dots (OPENEN, $ub + 25, r_5, ub + 25$)
 (PUBLICPERMUTE, $ub + 16, ub + 18, ub + 20, ub + 22, ub + 24, r^{-1}, ub + 30$)

To see that $SIM-\mathcal{IN}\mathcal{V}_\sigma$ is the required simulator for $\mathcal{IN}\mathcal{V}_\sigma$, consider a random execution e of $\mathcal{IN}\mathcal{V}_\sigma$ on gm^0 and a random output E of $SIM-\mathcal{IN}\mathcal{V}_\sigma(\text{ub}^0)$. 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:

- (OPENEN, $\text{ub}^0 + 17, \hat{\nu}_1, \text{ub}^0 + 25$) \dots (OPENEN, $\text{ub}^0 + 25, \hat{\nu}_5, \text{ub}^0 + 25$)
 (PUBLICPERMUTE, $\text{ub}^0 + 16, \text{ub}^0 + 18, \text{ub}^0 + 20, \text{ub}^0 + 22, \text{ub}^0 + 24, \hat{\nu}^{-1}, \text{ub}^0 + 30$)

Thus, while the last 6 records produced by e and E may very well be different, they are identically distributed. In fact, $r = r_1 r_2 r_3 r_4 r_5$ 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.*

6.2 A Ballot-Box Computer for Permutation Product

Public mediator $MULT_{\sigma, \tau}$

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

- (1) Execute computer \mathcal{INV}_{σ} to obtain the envelope encoding α .
- (2) For $\ell = 1$ to 5: make a new super-envelope A_{ℓ} containing the pair of envelopes $(\tau_{\ell}, \alpha_{\ell})$.
- (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 $\hat{\nu} = \hat{\nu}_1 \circ \dots \circ \hat{\nu}_5$.
- (6) Publicly permute μ according to $\hat{\nu}^{-1}$ to obtain ρ .

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

Lemma 2. *For any two, disjoint, 5-long addresses σ and τ , $MULT_{\sigma, \tau}$ 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 $MULT_{\sigma, \tau}$ on any initial memory gm^0 proper for permutation product and σ, τ , and let \mathbf{ub}^0 be the identifier upper-bound of gm^0 .

Correctness. Note that by Lemma 1, Step 1 generates 30 new identifiers (increasing \mathbf{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 ℓ th 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, $\rho = \mathbf{ub}^0 + 51, \dots, \mathbf{ub}^0 + 55$; 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 $SIM-MULT_{\sigma, \tau}$

Input: ub —an integer;

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

- (NEWSUP, $ub + 26, \tau_1, ub + 31$) \dots (NEWSUP, $ub + 30, \tau_5, ub + 35$)
(BALLOTBOX, $ub + 31, \dots, ub + 35, ub + 40$)
(OPENSUP, $ub + 36, ub + 42$) (OPENSUP, $ub + 37, ub + 44$) \dots (OPENSUP, $ub + 40, ub + 50$)
(OPENEN, $ub + 42, r_1, ub + 50$) (OPENEN, $ub + 44, r_2, ub + 50$) \dots (OPENEN, $ub + 50, r_5, ub + 50$)
(PUBLICPERMUTE, $ub + 41, ub + 43, ub + 45, ub + 47, ub + 49, r^{-1}, ub + 55$)

To see that $SIM-MULT_{\sigma,\tau}$ is the required simulator for $MULT_{\sigma,\tau}$, consider a random execution e of $MULT_{\sigma,\tau}$ on gm^0 and a random execution E of $SIM-MULT_{\sigma,\tau}(ub^0)$. Lemma 1 implies that the 22 records produced by protocol INV_{σ} in conceptual Step 1 of execution e are distributed identically to the first 22 produced by $SIM-invert_{\sigma}(ub^0)$ in E . Next, 11 records produced by Steps 2-4 of $MULT_{\sigma,\tau}$ 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 $SIM-INV_{\sigma}(ub^0)$ and of Step 2 of $SIM-MULT_{\sigma,\tau}(ub^0)$ are selected independently from each other. The same holds for (secret) permutations selected by the ballot-box in Steps 6 of INV_{σ} and of $MULT_{\sigma,\tau}$. Therefore, the whole records produced by e and E are identically distributed.

Clean Operation. Evident from our construction. *Q.E.D.*

6.3 A Ballot-Box Computer for Permutation Clone

Public mediator $CLON\mathcal{E}_{\sigma}$

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

- (1) Execute computer INV_{σ} 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, \dots, A_5 to obtain A'_1, \dots, 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 $\hat{\eta} = \hat{\eta}_1 \circ \dots \circ \hat{\eta}_5$.
- (7) Publicly permute μ and ν according to $\hat{\eta}^{-1}$.²⁰

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

Lemma 3. *For any 5-long address σ , $CLON\mathcal{E}_{\sigma}$ is a ballot-box computer for permutation clone, with input address σ and output addresses 66, \dots , 70 and 71, \dots , 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 $SIM-CLON\mathcal{E}_{\sigma}$

Input: ub —an integer.

- (1) Run $SIM-INV_{\sigma}(ub)$;
- (2) Randomly select a permutation $r \in \mathbb{S}_5$ and add to the output the following sequence of records:
 - (NEWEN, 1, $ub + 31$) \dots (NEWEN, 5, $ub + 35$) (NEWEN, 1, $ub + 36$) \dots (NEWEN, 5, $ub + 40$)
 - (NEWSUP, $ub + 26, ub + 31, ub + 36, ub + 41$) \dots (NEWSUP, $ub + 30, ub + 35, ub + 40, ub + 45$)
 - (BALLOTBOX, $ub + 41, \dots, ub + 45, ub + 50$)
 - (OPENSUP, $ub + 46, ub + 53$) (OPENSUP, $ub + 47, ub + 56$) \dots (OPENSUP, $ub + 50, ub + 65$)
 - (OPENEN, $ub + 53, r_1, ub + 65$) (OPENEN, $ub + 56, r_2, ub + 65$) \dots
 (OPENEN, $ub + 65, r_5, ub + 65$)
 - (PUBLICPERMUTE, $ub + 51, ub + 54, ub + 57, ub + 60, ub + 63, r^{-1}, ub + 70$)
 (PUBLICPERMUTE, $ub + 52, ub + 55, ub + 58, ub + 61, ub + 64, r^{-1}, ub + 75$)

Q.E.D.

²⁰Note that Steps 2–7, in essence, correspond to a protocol for permutation inverse that on input α produces two identical envelope encodings, each encoding $\hat{\alpha}^{-1}$.

7 General Ballot-Box Computers

Without loss of generality, any finite function can be represented as a binary function $f : \{0, 1\}^a \rightarrow \{0, 1\}^b$, and it is well known in theoretical computer science that any such f can be represented as a fixed sequence of the following basic functions:

- *COIN*, the probabilistic function that, on no input, returns a random bit;
- *DUPLICATE*, the function that, on input a bit b , returns a pair of bits (b, b) ;
- *AND*, the function that, on input a pair of bits (b_1, b_2) , returns 1 if and only if $b_1 = b_2 = 1$; and
- *NOT*, the function that, on input a bit b , returns the bit $1 - b$.

We thus intend to prove that each of these basic functions has a ballot-box computer, and then obtain a ballot-box computer for any desired $f : \{0, 1\}^a \rightarrow \{0, 1\}^b$ by utilizing these 4 basic computers. To this end, we must first decide how to encode binary strings and binary functions.

Definition 8. We define the \mathbb{S}_5 encoding of a k -bit binary string $z = b_1, \dots, b_k$, denoted by \bar{z} , to be $\bar{b}_1 \cdots \bar{b}_k$, where

$$\bar{0} = 12345; \quad \bar{1} = 12453.$$

The \mathbb{S}_5 encoding is immediately extended to binary functions as follows: $\bar{f}(\bar{z}) = \overline{f(z)}$ for all z .

One of our basic computer has already been constructed: namely,

Lemma 4. For any envelope encoding σ , \mathcal{CLONE} is a ballot-box computer for $\overline{\text{DUPLICATE}}$ with input address σ .

Proof. Because \mathcal{CLONE} duplicates any permutation in \mathbb{S}_5 , in particular it duplicates 12345 and 12453.
Q.E.D.

We thus proceed to build the other 3 basic computers.

7.1 A Ballot-Box Computer for $\overline{\text{COIN}}$

Public mediator \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 $\alpha_1, \dots, \alpha_5$ and β_1, \dots, β_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. \mathcal{COIN} is a ballot-box computer for $\overline{\text{COIN}}$, with no input address and output address 15, ..., 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-COIN

Input: ub —an integer.

Output the following sequence of records:

(NEWEN, 1, $ub + 1$) (NEWEN, 2, $ub + 2$) (NEWEN, 3, $ub + 3$) (NEWEN, 4, $ub + 4$) (NEWEN, 5, $ub + 5$)
 (NEWEN, 1, $ub + 6$) (NEWEN, 2, $ub + 7$) (NEWEN, 4, $ub + 8$) (NEWEN, 5, $ub + 9$) (NEWEN, 3, $ub + 10$)
 (NEWSUP, 1, 2, 3, 4, 5, $ub + 11$) (NEWSUP, 6, 7, 8, 9, 10, $ub + 12$)
 (BALLOTBOX, 11, 12, $ub + 14$) (OPENSUP, 13, $ub + 14$) (DESTROY, 14, $ub + 14$)

Q.E.D.

7.2 Ballot-Box Computers for \overline{NOT} and \overline{AND}

In proving the existence of ballot-box computers for \overline{NOT} and \overline{AND} we rely on the result of Barrington (1986) that the Boolean functions NOT and AND can be realized as sequences of group operations in \mathbb{S}_5 .²¹ Here is our rendition of it.

Let $\mathcal{I} = 12345$, $a = 12453$, $b = 25341$, $c_1 = 34125$, $c_2 = 12354$, and $c_3 = 42153$; and let \tilde{x} , x' and x^* be the operators defined to act on a permutation $x \in \mathbb{S}_5$ as follows:

$$\tilde{x} = c_1^{-1}xc_1, \quad x' = c_2^{-1}xc_2, \quad \text{and} \quad x^* = c_3^{-1}xc_3.$$

Then, recalling that $\bar{0} = \mathcal{I}$ and $\bar{1} = a$, the following lemma can be verified by direct inspection.

Barrington's Lemma. *If $x_1 = \bar{b}_1$ and $x_2 = \bar{b}_2$, where b_1 and b_2 are bits, then*

$$\overline{\bar{b}_1} = (x_1a^{-1})^*, \quad \text{and} \quad \overline{b_1 \wedge b_2} = (x_1\tilde{x}_2x_1^{-1}\tilde{x}_2^{-1})'.$$

Lemma 6. *There exist ballot-box computers \mathcal{NOT} and \mathcal{AND} for, respectively, \overline{NOT} and \overline{AND} .*

Proof. The lemma follows by combining Barrington's Lemma 6 and our Lemmas 1,2 and 3. That is, each of \mathcal{NOT} and \mathcal{AND} is obtained in four steps. First, by expanding the operators of the formulas of Lemma 6 so as to show all relevant constants a , c_1 , c_2 and c_3 . Second, by generating envelope encodings for each occurrence of each constant. Third, in the case of \mathcal{AND} , by using our elementary computer \mathcal{CLONE} so as to duplicate x_1 and x_2 . Forth, by replacing each occurrence of permutation inverse and permutation product in the formulas of Lemma 6 with, respectively, our elementary computers \mathcal{INV} and \mathcal{MULT} . Accordingly, the simulators for \mathcal{NOT} and \mathcal{AND} can be obtained by running the simulators of their individual elementary computers in the proper order (feeding them as input the right values of ub).

Q.E.D.

7.3 Ballot-Box Computers for Arbitrary Finite Functions

Theorem 1. *For any finite function f there exists a ballot-box computer \mathcal{P}_f for f .*

Proof. Basic results of complexity theory guarantee that for any finite binary function $f : \{0, 1\}^a \rightarrow \{0, 1\}^b$, there exists a fixed sequence $C_f = F_1, F_2, \dots, F_K$ such that:

²¹ Note that neither *DUPLICATE* nor *COIN* can be realized in \mathbb{S}_5 , and thus one cannot compute arbitrary functions in \mathbb{S}_5 . Indeed, the result of Barrington (1986) was solely concerned with implementing a restricted class of finite functions called NC^1 . 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.

- Each F_i is either *COIN*, *DUPLICATE*, *NOT* or *AND*;
- Each input bit of F_i is either one of the original input bits or one of the output bits of F_j for $j < i$; and
- For each a -bit input x , the b -bit output $f(x)$ can be computed by evaluating (in order) all functions F_i on their proper inputs, and then concatenating (in order) all their output bits not used as inputs by some F_j . (Such bits are guaranteed to be exactly b in total.)

Define now \mathcal{P}_i as follows:

$\mathcal{P}_i = \text{COIN}$ if $F_i = \text{COIN}$;

$\mathcal{P}_i = \text{CLON}\mathcal{E}$ if $F_i = \text{DUPLICATE}$;

$\mathcal{P}_i = \text{NOT}$ if $F_i = \text{NOT}$;

$\mathcal{P}_i = \text{AND}$ if $F_i = \text{AND}$.

Let \mathcal{P} be the concatenation of $\mathcal{P}_1, \dots, \mathcal{P}_K$, with appropriately chosen input addresses, so that the l th input address of \mathcal{P}_i matches the m th output address of \mathcal{P}_j whenever the l th input bit of F_i is the m th output bit of F_j . Then, \mathcal{P} is a ballot-box computer for f . In fact, \mathcal{P} 's correctness follows from the correctness of each computer \mathcal{P}_i . Furthermore, \mathcal{P} 's privacy follows from the fact that each \mathcal{P}_i has a simulator SIM_i , and thus a simulator SIM for \mathcal{P} can be obtained by executing (in order) $SIM_1, SIM_2, \dots, SIM_K$. Specifically, on input an identifier upper-bound ub_0 , SIM runs $SIM_1(\text{ub}_0)$ to generate a public record R_1 , computes the identifier upper-bound ub_1 of R_1 , runs $SIM_2(\text{ub}_1)$ to obtain a public record R_2 and computes ub_2 , and so on.²² At the end, SIM returns the public record $R = R_1 \circ R_2 \circ \dots \circ R_K$. Finally, the clean operation of \mathcal{P} follows from the clean operation of each \mathcal{P}_i . *Q.E.D.*

For completeness, we explicitly exhibit the sequence C_f and our computer \mathcal{P} in, respectively, Appendices A and B.

Remarks.

- Note that our ballot-box computer \mathcal{P}_f is “as efficient as” f itself. Indeed, the description of \mathcal{P}_f is linear in the description of C_f . This is so because, letting $C_f = F_1, F_2, \dots$, \mathcal{P}_f replaces each F_i with a ballot-box computer for $\overline{F_i}$ that has constant number of actions and generates a constant number of identifiers. Moreover, assuming each function F_i is executable in constant time (since it operates on at most two bits) and assuming that each ballot-box action is executable in constant time (since it operates on at most 5 ballots), the time needed to run \mathcal{P}_f is also linear in the time needed to run C_f .
- If \mathcal{P} is executed on an initial global memory whose ballot set coincides with just the input envelopes, then the ballot set of \mathcal{P} 's final global memory coincides with just the output envelopes.

8 Public Checking of Inputs

So far, we have assumed that a ballot-box computer P_f for function f is executed starting on a proper global memories. That is, we have assumed that P_f 's input envelopes indeed encode “legitimate” inputs for f . Ultimately, however, P_f will be run on a global memory consisting of n sequences of envelopes, where the i th sequence has been prepared by player i so as to encode his chosen input to f . Specifically, if f 's inputs are L -bit strings, i is supposed to make $5L$ envelopes containing the \mathbb{S}_5 encoding of his chosen string. However, there is no guarantee that player i will indeed so prepare his sequence of envelopes. Accordingly, we must construct a special public ballot-box mediator, the *public input-checker*, that, without revealing any undue information, verifies whether or not a sequence of $5L$ envelopes contains the \mathbb{S}_5 encoding of an L -bit string.

Such a checker \mathcal{C} replaces a sequence of $5L$ envelopes E with a new $5L$ -envelope sequence E' so that:

²²Notice that there are two ways of computing ub_i . The first is “extracting” ub_i from R_i . The second, is to add a specific constant to ub_{i-1} , since each of *COIN*, *CLON* \mathcal{E} , *NOT*, and *AND* generates a fixed number of new identifiers.

- If E contains the \mathbb{S}_5 encoding of an L -bit string s , then so does E' .²³
- Else, E' consists of $5L$ envelopes, publicly made so as to contain the \mathbb{S}_5 encoding of 0^L .

In other words, \mathcal{C} must be consistent with our way of handling aborts. Let us now see how to formalize \mathcal{C} 's correctness and privacy.

For \mathcal{C} to be a public mediator, its correctness should be ascertainable from the public record $R_{\mathcal{C}}$ it generates. We thus demand that there exists a pre-specified function Ch —for checking—such that, $Ch(R_{\mathcal{C}}) = 0$ if the content of E is an \mathbb{S}_5 encoding of an L -bit string, and 1 otherwise.

Let us now focus on \mathcal{C} 's privacy requirements. If E correctly encodes an L -bit string z , then no information—about z or anything else—should be revealed. (As usual, we formalize this no-additional-information requirement via a simulator that, just given the current identifier upper-bound, generates a public record identically distributed as the one generated by \mathcal{C} .) Else, it does not matter what might be revealed about the content of E :²⁴ the player has chosen to abort and his input is publicly reset to 0^L .

The final condition is that checking for the validity of a player's L -bit input should not alter (or reveal information about) the inputs of the other players. This is formalized by the “clean operation” property, which implies that in checking the input envelopes of player i no other existing envelope is ever “touched.”²⁵

Definition 9. Let x be a $5L$ -long address and $\mathcal{C} = C^1, \dots, C^K$ a public ballot-box mediator. We say that \mathcal{C} is an L -bit input checker with input address x if there exist a simulator SIM , a checking predicate Ch , and a $5L$ -long address w such that, for any execution e of \mathcal{C} on an initial memory $gm^0 = (B^0, R^0, H^0)$ —with identifier upper-bound \mathbf{ub} and $I_x \subset I_{B^0}$ —the following three properties hold:

1. Correctness: If $\text{cont}_{B^0}(x) = \bar{z}$ for some $z \in \{0, 1\}^L$, then $Ch(R_{\mathcal{C}}^K(e)) = 1$ and $\text{cont}_{B^{\kappa}(e)}(w + \mathbf{ub}) = \bar{z}$.
Else, $Ch(R_{\mathcal{C}}^K(e)) = 0$ and $\text{cont}_{B^{\kappa}(e)}(w + \mathbf{ub}) = \overline{0^L}$.
2. Privacy: If $\text{cont}_{B^0}(x) = \bar{z}$ for some $z \in \{0, 1\}^L$, then $R_{\mathcal{C}}^K(e) = SIM(\mathbf{ub})$.
3. Clean Operation: $B^K(e) = B_{\{w+\mathbf{ub}\}} \cup B^0 \setminus B_{\{x\}}$.

Lemma 7. For any L , there exists an L -bit input checker.

Proof. Let \mathcal{C}_1 be the public ballot-box mediator that, given an envelope encoding x of a bit b , operates as follows:

1. run \mathcal{CLONE} twice to generate three envelope encodings of b , namely, σ, β, γ ;
2. run \mathcal{NOT} on β to obtain the envelope encoding δ of the complementary bit, $1 - b$;
3. create super-envelopes A and B containing, respectively, the envelope sequence γ and the envelope sequence δ ;
4. ballot-box A and B to obtain super-envelopes C and D ;
5. open C and D to reveal, respectively, a sequence of 5 envelopes ρ and κ ; and
6. open all the envelopes of ρ and κ .

Then notice that the contents revealed in step 6 are, with equal probability, either (\mathcal{I}, a) or (a, \mathcal{I}) . Notice too, that \mathcal{C}_1 is perfectly simulatable, and thus does not reveal anything about b .

Consider now executing \mathcal{C}_1 when x is the encoding of a permutation $p \in \mathbb{S}_5$ that is neither $\mathcal{I} = 12345$ nor $a = 12453$. Then, in Step 6, ρ and κ will continue to be envelope encoding of permutations of \mathbb{S}_5 , but neither $\hat{\rho}$ nor $\hat{\kappa}$ will equal \mathcal{I} or a . Indeed, recall that the Barrington function for \mathcal{NOT} , $f(x) = (xa^{-1})^*$, is a bijection from \mathbb{S}_5 to \mathbb{S}_5 . Therefore, $f(x) \in \{\mathcal{I}, a\}$ if and only if $x \in \{\mathcal{I}, a\}$. Thus, because our ballot-box mediator \mathcal{NOT} “literally translates” $f(x)$, it will generate a envelope encoding of a bit if and only if

²³Ideally, \mathcal{C} would leave E intact after verifying that it has a legitimate content, but the ballot-box actions needed for such verification end up “destroying” the original envelopes.

²⁴What will be revealed will depend on the actual contents of E chosen by the player as well as on the way in which \mathcal{C} operates.

²⁵Indeed, any ballot-box action accessing the content of an envelope j removes j , for ever, from the set of ballot identifiers.

it starts with the envelope encoding of a bit. Thus, revealing $\hat{\rho}$ and $\hat{\kappa}$ in Step 6 proves that x was not the encoding of a bit.

Finally, consider executing \mathcal{C}_1 when x is not the encoding of a permutation in \mathbb{S}_5 . In this case, when executing the first conceptual step of \mathcal{CLONE} (i.e., when executing \mathcal{INV}), the original envelopes of x will be first randomly permuted and then opened, publicly proving that $\text{cont}(x) \notin \mathbb{S}_5$.

Therefore, constructing an L -bit input checker is rather simple. Given an address vector $x = x_1, \dots, x_L$, one runs \mathcal{C}_1 on each x_i . If for each i , x_i is revealed to be an envelope encoding of a bit, the input checker generates for each i envelopes σ_i , which content is guaranteed to be identical to the content of envelopes x_i . If for some i , it is revealed that the content of x_i were different from \mathcal{I} or a , then the input checker should destroy any envelopes it had generated so far and publicly generate L envelope encodings of \mathcal{I} .

Since the content of all opened envelopes becomes part of the public record, the required predicate Ch is trivially constructed. It evaluates to 0 when it becomes clear that for some i x_i is not a permutation or a permutation different from \mathcal{I} and a ; and to 1, otherwise. The required simulator SIM is also trivially obtained by executing, in appropriate order, simulators of the corresponding computers for \mathcal{CLONE} and \mathcal{NOT} and generating records of steps 3-6 of each \mathcal{C}_1 in a manner analogous to that of our previously constructed simulators (e.g. $SIM-MULT.$) *Q.E.D.*

9 Perfect Ballot-Box Implementations

In this Section we finally show that any normal-form mechanism has a perfect implementation. We do so in three steps. First, as discusses in Section 1.3., we augment the notion of a normal-form mechanism with “aborts.” Then, we formalize the notion of a perfect implementation. Lastly, we show that perfect implementations always exist.

9.1 Augmented Normal-form Mechanisms

Let us start by formally recalling traditional normal-form mechanisms.

Definition 10. 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^* , $M = M_1 \times \dots \times M_n$;
- Y , the outcome set, is a subset of Σ^* ; and
- g , the outcome function, is a probabilistic function, $g : M \rightarrow Y$.

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.

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

Notice that, without any loss of generality, we can focus on just standard normal-form mechanisms.²⁶ We now include aborts into standard normal-form mechanisms.

Definition 11. We say that a normal-form mechanism $\mathcal{M} = (N, M, Y, \hat{g})$ is an L -bit augmented normal-form mechanism if there exists a standard normal-form mechanism $(N, (\{0, 1\}^L)^n, \{0, 1\}^L, g)$ such that

- For each i , $M_i = \{0, 1\}^L \cup \{\text{abort}\}$, and

²⁶ 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, \dots, 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 $g' : (\{0, 1\}^L)^n \rightarrow \{0, 1\}^L$ 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 $g'(x'_1, \dots, x'_n) = y'$ if and only if $g(x_1, \dots, x_n) = y$. Then, $\mathcal{M} = (N, (\{0, 1\}^L)^n, \{0, 1\}^L, g')$ is a standard normal-form mechanism equivalent to \mathcal{M} .

- $Y = (\{0, 1\}^L) \times \{0, 1\}^N$.

The outcome function $\hat{g} : M \rightarrow Y$ of \mathcal{M} is defined via g as follows:

$$\hat{g}(m_1, \dots, m_n) = (g(m'_1, \dots, m'_n), (b_1, \dots, b_n)),$$

where for each i we have: $m'_i = m_i$ and $b_i = 0$ if $m_i \in \{0, 1\}^L$, and $m'_i = 0^L$ and $b_i = 1$ otherwise.

The choice of 0^L as the message of an aborting player is without loss of generality. We could have chosen any fixed element (or a random element) of $\{0, 1\}^L$ instead.

9.2 The Notion of a Perfect Implementation

We present our notion of a perfect implementation so as to be independent of ballots and ballot-boxes. That is, we present it for general implementation models of imperfect information, where the set of players includes a mediator and Nature.

Definition 12. An implementation model is a triple $\text{IM} = (N^*, \mathcal{H}, \mathcal{A})$, where:

- $N^* = N \cup \{\mathfrak{M}, \mathfrak{N}\}$ is a finite set of players, where each of the n members of N is referred to as an ordinary player, \mathfrak{M} as the mediator, and \mathfrak{N} as Nature.
- $\mathcal{H} = \mathcal{H}^0 \cup_{i \in N^*} \mathcal{H}^i$ is the set of histories. A member H^0 of \mathcal{H}^0 is referred to as a public history, and a member H^i of \mathcal{H}^i as the private history of player i . Each history is a sequence of strings, called records.
- \mathcal{A} is the set of actions. Each action of \mathcal{A} appends a new record (possibly empty) to each of the current $n + 3$ histories.

A perfect implementation is an extensive-form mechanism of imperfect information. Such a mechanism (see Osborne and Rubinstein (1997) for a formal definition) is a game form of an extensive-form game G , that is, G without preferences over the termination nodes and initial moves of Nature assigning types to the players. (Of course, for equilibrium analysis both must be added back.) Accordingly, Nature is a special player that, at each of its decision nodes, chooses its actions according to a pre-specified and commonly known distribution. By convention, we think of Nature's private history as encoding the "state of the world."

Definition 13. Let $\mathcal{M} = (N, M, Y, \hat{g})$ be an augmented normal-form mechanism and \mathcal{M}' an extensive-form mechanism, in an implementation model $\text{IM} = (N^*, \mathcal{H}, \mathcal{A})$, played starting with empty histories. We say that \mathcal{M}' is a perfect implementation of \mathcal{M} if the following properties are satisfied:

- Public Outcome. The outcome of \mathcal{M}' is a pre-specified function of the final public history H^0 .
- Public Mediation. The private history of \mathfrak{M} is always empty, and at each of \mathfrak{M} 's decision nodes there is only one available action, determined by a pre-specified function of the then current H^0 .
- Strategy and Privacy Equivalence. Letting (N, S', Y', g') be the normal form representation of \mathcal{M}' ,²⁷

1. $Y' = Y$

2. For all $i \in N$, $S'_i = M'_i \cup \{A_i\}$

(Comment: as we shall see, A_i represents the set of all "aborting" strategies of i in \mathcal{M}')

For all $i \in N$, there exists $\bar{m}_i \in M'_i$ such that, arbitrarily fixing $\bar{s}_i \in A_i$

3. For all $s_i \in A_i$ and all $s_{-i} \in S'_{-i}$, $g'(s_i, s_{-i}) = g'(\bar{s}_i, s_{-i})$

(Comment: all aborting strategies of i in \mathcal{M}' are outcome-equivalent to each other)

4. There exists a bijection $\psi_i : M_i \leftrightarrow M'_i \cup \{\bar{s}_i\}$ such that $\psi_i(\text{abort}) = \bar{s}_i$, $\psi_i(0^L) = \bar{m}_i$, and for all $(m_1, \dots, m_n) \in M$, $\hat{g}(m_1, \dots, m_n) = g'(\psi_1(m_1), \dots, \psi_n(m_n))$

²⁷The public mediator and Nature do not enter the set of players of the normal-form representation as they have no choice in performing their actions and no preferences defined over final outcomes.

5. There exists a probabilistic function SIM such that, for any subset $C \subset N$, any strategy profile $s = (s_C, s_{-C})$ such that $s_{-C} \in M'_{-C}$, and any $y \in Y'$, in a random execution of \mathcal{M}' with strategy profile s and outcome y , the vector of final histories (H^0, H^C) is distributed as $SIM(C, s_C, y)$.

(Comment: This expresses that, provided that corresponding strategies are used and the same outcome obtained, the information about the other players collectively available to the members of C after a play of \mathcal{M}' and after a play of \mathcal{M} coincide, provided that the players in $-C$ use non-aborting strategies. If a player $i \in -C$ aborts, then his identity is part of y in any case, and we do not care whether information about his specific aborting strategy is revealed.)

9.3 The Existence of Perfect Ballot-Box Implementations

Let us now transform the ballot-box model of Section 3 into the following implementation model $BB = (N^*, \mathcal{H}, \mathcal{A})$. In Section 4, each global memory gm consists of the current set of ballots B and public record R . In BB , the public-record sequence R is simply identified as the public history H^0 , and B is considered part of \mathfrak{N} 's private record. All ballot-box actions continue to be actions in BB . Specifically, the 7 classes of public actions are reserved to \mathfrak{M} , and actions of Nature are reserved to \mathfrak{N} . (Thus, when \mathfrak{N} ballot-boxes a sequence of K ballots, the actual permutation $\pi \in \mathbb{S}_K$ is appended to $H^{\mathfrak{N}}$.) Finally, we complete our specification of BB by adding the following class of actions for the n players in N .

$(SIMPRIVEN, L_1, c_1), \dots, (SIMPRIVEN, L_n, c_n)$ —where $c_i \in \Sigma^{L_i}$.

“Each $i \in N$ simultaneously and privately makes a sequence of L_i envelopes (with contents c_i).”

$B := B \cup \{(\mathbf{ub} + 1, c_1^1, 0), \dots, (\mathbf{ub} + L_1 + \dots + L_n, c_n^{L_n})\}$; $\mathbf{ub} := \mathbf{ub} + L_1 + \dots + L_n$;

$H^1 := c_1; \dots; H^n := c_n$; and $H^0 := H^0 \circ (SIMPRIVEN, L_1, \dots, L_n, \mathbf{ub})$.

We refer to the so specified BB as *the ballot-box implementation model*.

Theorem 2. *Every L -bit augmented normal-form mechanism $\mathcal{M} = (N, M, Y, \hat{g})$ has a perfect implementation in BB .*

Proof. Consider the following extensive-form mechanism \mathcal{B} :

(Initialization.) For each $i \in N$, the bit variable b_i is set to 0.

1. Each player $i \in N$, simultaneously and privately, makes a sequence E_i of envelopes.
2. If i did not make $5L$ envelopes, then the mediator (1) destroys all of i 's envelopes, if any, (2) publicly generates a sequence of envelopes E_i consisting of L envelope encodings of 0, and (3) sets $b_i = 1$.
3. For each $i \in N$, the public mediator (1) runs the L -bit checker constructed in Lemma 7 on the sequence E_i , so as to generate a sequence E'_i of $5L$ envelopes, and (2) if $b_i = 0$, sets b_i to the value of the checking predicate.
4. The public mediator executes a ballot-box computer P_g for the function g on input envelopes E'_1, \dots, E'_n , so as to obtain a sequence of output envelopes T_1, \dots, T_L .
5. The public mediator publicly opens all output envelopes so as to reveal an L -bit string y .
6. The outcome of \mathcal{B} is the \mathbb{S}_5 decoding of y and the n -bit vector (b_1, \dots, b_n) .

Let us now prove that \mathcal{B} is a perfect implementation of \mathcal{M} .

First for all, note that each player $i \in N$ acts only once, in Step 1, when his information set is “empty.” Thus, the set of i 's strategies S_i consists of all possible actions $(SIMPRIVEN, L_i, c_i)$. We now partition S_i into M'_i and A_i as follows: a strategy belongs to M'_i if and only if it coincides with the action $(SIMPRIVEN, 5L, c)$, where c is the Barrington encoding of some $z \in \{0, 1\}^L$. For any $z \in \{0, 1\}^L$, we refer to this strategy as $s_{i,z}$. After an arbitrary selection of $\bar{s}_i \in A_i$ and setting $\bar{m}_i = s_{i,0^L}$, we define the bijection ψ_i as follows: $\psi_i(z) = s_{i,z}$ for all $z \in \{0, 1\}^L$ and $\psi_i(\text{abort}) = \bar{s}_i$.

For each player i , at the end of Step 3 and independently of the strategies of the other players, a strategy $s_i \in A_i$ leads to $b_i = 1$ and $\text{cont}(E'_i) = \overline{0^L}$. (Indeed, if s_i does not make $5L$ envelopes, b_i is set to 1 in Step 2, and the mediator publicly generates a sequence E_i of $5L$ envelopes consisting of L envelope encodings of 0. Thus, when the input checker is run on such E_i , it generates a new sequence E'_i whose contents are the Barrington encoding of 0^L , and keeps $b_i = 1$. Else, if i makes exactly $5L$ envelopes, but their contents is not the \mathbb{S}_5 encoding of an L -bit string, then in Step 3 the input checker sets for the first time $b_i = 1$ and again generates a sequence of envelopes E'_i with contents $\overline{0^L}$.) If, instead, i plays a strategy $s_{i,z} \in M'_i$, then by the correctness of the input checker, at the end of Step 3 $b_i = 0$ and the content of E'_i is \bar{z} . In turn, for all players, the correctness property of computer P_g implies that the content of envelopes T coincides with $g(\text{cont}(E'_1), \dots, \text{cont}(E'_n))$. Therefore: (a) Property 1 of Definition 13 is trivially established; (b) any strategy $s_i \in A_i$ leads to the same outcome as any other aborting strategy, which establishes Property 3; and (c) for any strategy profile s so that, for all i , $s_i \in M'_i \cup \{\bar{s}_i\}$, the outcome $g'(s)$ of \mathcal{B} equals $\hat{g}(\psi_1^{-1}(s_1), \dots, \psi_n^{-1}(s_n))$, which establishes Property 4.

Finally, let us establish Property 5. Arbitrarily fix a subset $C \subset N$, a strategy profile $s = (s_C, s_{-C})$ such that $s_{-C} \in M'_{-C}$, and a possible outcome y for a play of \mathcal{B} with s . Note that the private histories H^C deterministically depend on s^C alone. (Indeed, for each $i \in C$, the private history H^i in a play of \mathcal{B} consists of the content of the envelopes made by i .) Accordingly, the existence of an algorithm SIM that, on input s^C , outputs the correct sub-profile of histories H^C is trivial. The only difficulty is to establish the existence of a probabilistic SIM that, on input y , outputs H^0 according the proper distribution. We proceed in a case-by-case basis. If $y = (v, (0, \dots, 0))$ —i.e., if no player aborts— then SIM probabilistically generates H^0 by running (n times) the simulator of the input checker, then running the simulator of the ballot-box computer P^f , and, finally, appending to H^0 a sequence of OPENEN records with revealed contents being the \mathbb{S}_5 encoding of v . Assume now that, for some $i \in C$, $b_i = 1$. Then, part of H^0 also depends on s_i . Letting $s_i = (\text{SIMPRIVEN}, L_i, c_i)$ we distinguish two cases: (1) $L_i \neq 5L$, and (2) $L_i = 5L$ and $c_i \neq \bar{z}$ for some $z \in \{0, 1\}^L$. In case 1, the public record of Step 1 contains L_i , and the public records appended to H^0 in Step 2 are a fixed string of $L_i + 5L$ records (namely, L_i destroy-envelope records and $5L$ new-envelope records). Clearly SIM can generate these records on input s_i , or even L_i only. In case 2, the specified L -bit checker, for alleged envelope-encoding of i , reveals a random permutation of the 5 strings contained in the corresponding 5 envelopes. Since these 5 strings are part of s_i (indeed part of just c_i), SIM can trivially randomly permute them. In both cases, the rest of H^0 is generated (with the right odds) by the simulators of the input checkers and computer P^f .

Q.E.D.

10 Final Remarks

The Universality of Our 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 $p(q(i))$.

Complementing Mechanism Design. The practical meaningfulness of any abstraction depends on whether concrete implementations that adequately approximate it exist. In a sense, therefore, our con-

tributions 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.²⁸ Thus, while we do not help a designer in engineering new mechanisms, by perfectly implementing whatever abstract mechanisms he finds, we do enable him to ignore issues of privacy and trust in his work.

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.²⁹ 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 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

A Finite Functions as 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, \dots, 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, \dots, 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 $\text{maxheight}(u)$, is the maximum integer k for which there exists a path s_1, \dots, s_k in which s_1 is a source and $s_k = u$.

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

²⁸Sjöström 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.

²⁹For 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.

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 $maxheight \geq 3$ 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.

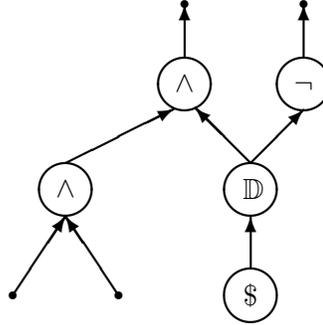


Figure 1: Graphical representation of a Boolean circuit with two input nodes.

Interpreting 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 $\mathcal{F}_C : \{0, 1\}^a \rightarrow \{0, 1\}^b$ 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 i th 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 $\neg b$;³⁰
 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.

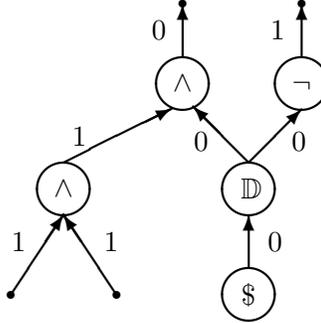


Figure 2: A computation of the Boolean circuit of Figure 1, on input 11 and coin toss 0.

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 .

B Constructing Ballot-Box Computers for Arbitrary 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 public ballot-box mediator \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 $Func(j)$ 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) NOT , AND , $CLONE$, and $COIN$, 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 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 ub^0 from R^0 .) Algorithm \mathbb{P} constructs \mathcal{P} as the concatenation of ballot-box computers, one for each computation node of C . If j is such a computational node, the corresponding public ballot-box mediator 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;
 $x = x_1, \dots, x_a$ —an address such that each x_i has length 5.

- (1) Set $w_{a+1} = ub^0$.

³⁰Note: 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.

- (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 $\mathcal{P}_i = COIN$; label i 's outgoing edge with $\sigma_i = 15 + w_i, \dots, 19 + w_i$; and set $w_{i+1} = w_i + 19$;
 - if $Func(i) = NOT$ and σ labels i 's incoming edge
 - then set $\mathcal{P}_i = NOT_\sigma$; label i 's outgoing edge with $\sigma_i = 61 + w_i, \dots, 65 + w_i$; and set $w_{i+1} = w_i + 65$;
 - if $Func(i) = AND$ and $(\sigma; \tau)$ label i 's incoming edges,
 - then set $\mathcal{P}_i = AND_{\sigma, \tau}$; label i 's outgoing edge with $\sigma_i = 451 + w_i, \dots, 455 + w_i$; set $w_{i+1} = w_i + 455$;
 - if $Func(i) = DUPLICATE$ and σ labels i 's incoming edge,
 - then set $\mathcal{P}_i = CLONE_\sigma$; label i 's outgoing edges with $\sigma_i = 61 + w_i, \dots, 65 + w_i$ and $\tau_i = 66 + w_i, \dots, 70 + w_i$; and set $w_{i+1} = w_i + 70$;
- (4) For $i = 1$ to b : if α labels the incoming edge of sink $a + c + i$, then set $y_i = \alpha - w^0$;
- (5) Set $\mathcal{P}_C = \mathcal{P}_{a+1} \circ \dots \circ \mathcal{P}_{a+c}$.

Outputs: \mathcal{P}_C —a public ballot-box mediator; and
 $y = y_1, \dots, y_b$ —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 $x = x_1, \dots, x_a$ an address such that each x_i has length 5, then public ballot-box mediator \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 ub^0 is the identifier upper bound of the initial memory gm^0 , proper for \mathcal{F}_C and $x = x_1 \dots x_a$, then for each computation node j , w_j is the initial identifier upper-bound at the start of execution of \mathcal{P}_j . Indeed, $w_{a+1} = ub^0$, 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 NOT always increases ub by 65, AND by 455, $CLONE$ by 70, and $COIN$ by 19.) Second, \mathbb{P} labels each edge with 5 distinct integers. Third, due to the correctness of NOT , AND , $CLONE$, and $COIN$ 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 $SIM-\mathcal{P}_C$ is the required simulator for \mathcal{P}_C .

$SIM-\mathcal{P}_C$

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 $SIM-\mathcal{P}_i(ub_i)$.

Clean Operation trivially holds for \mathcal{P}_C as it holds for each individual ballot-box computer \mathcal{P}_j . *Q.E.D.*

C 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, \dots, 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 the “public version” \mathcal{M}^* of \mathcal{M} , that maps (m_1, \dots, m_n) to a public outcome

with $n + 1$ components: namely, (y, y_1, \dots, y_n) , and its perfect ballot-box implementation \mathcal{B} constructed in Theorem 2. By Step 5, \mathcal{B} produces $n + 1$ sequences of envelopes, whose contents respectively encode y, y_1, \dots, y_n in a private and correct manner. Thus, to perfectly implement the original \mathcal{M} , one only needs to modify Step 5 of \mathcal{B} , so that (1) the public mediator 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 .

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

1. Definition 10 of a Normal-Form Mechanism is modified as follows:

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

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

2. The following class of ballot-box actions to be performed by a public mediator is added:

(SIMHANDOUT, L_1, \dots, L_n) —where $\forall i \in N$, $L_i \subset I_B$ and $\forall i \neq j$, $L_i \cap L_j = \emptyset$.

“For each $i \in N$, simultaneously and privately hand out envelopes L_i to player i .”

$B := B \setminus \{B_{L_1} \cup \dots \cup B_{L_n}\}$; $H^1 := H^1 \circ \text{cont}_B(L_1), \dots, H^n := H^n \circ \text{cont}_B(L_n)$; and $H^0 := H^0 \circ (\text{SIMHANDOUT}), L_1, \dots, L_n, \mathbf{ub}$.

3. The only change required to the notion of perfect implementation is adding the private inputs y^C to the set of arguments of the simulator SIM in Property 5 of Definition 12 .

4. The mechanism and the proof of Theorem 2 are modified as follows. In Step 5 of the mechanism, in addition to publicly opening the envelopes containing public output, the public mediator also performs (SIMHANDOUT, L_1, \dots, L_n) action, that is, privately distributes to each player i a sequence of envelopes L_i , who then privately reads the contents of L_i to obtain his own private output y_i . Note that the only addition to the public history H^0 is the record of (SIMHANDOUT) action which is fixed and can be trivially simulated by SIM . The profile of private histories H^C now also contains the contents of envelopes L_i for each $i \in C$. But since these contents are guaranteed to be the \mathbb{S}_5 encoding of y_i by correctness of \mathcal{P}_g , they also can be trivially simulated by SIM .

References

- Aumann, R. J. and Hart, S.: 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 , *Proceedings of the 18th Symposium on Theory of Computing*, ACM, pp. 1–5.
- Ben-Or, M., Goldwasser, S. and Wigderson, A.: 1988, Completeness theorems for fault-tolerant distributed computing, *Proceedings of the 20th Symposium on Theory of Computing*, ACM, pp. 1–10.
- 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.
- Ben-Porath, E.: 2003, Cheap talk in games with incomplete information, *Journal of Economic Theory* **108**(1), 45–71.
- Bolton, P. and Dewatripont, M.: 2005, *Contract Theory*, MIT Press, Cambridge, Massachusetts.

- Brandt, F. and Sandholm, T.: 2004, (Im)possibility of unconditionally privacy-preserving auctions, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, IEEE, pp. 810–817.
- Chaum, D., Crépeau, C. and Damgård, I.: 1988, Multi-party unconditionally secure protocols, *Proceedings of the 20th Symposium on Theory of Computing*, ACM, pp. 11–19.
- Agrawal, D., Archambeault, B., Rao, J.R., and Rohatgi, P.: 2002, The EM side-channel(s), *Proceedings of the 20th Symposium on Theory of Computing, Cryptographic Hardware and Embedded Systems Conference (CHES '02)*.
- Dodis, Y., Halevi, S. and Rabin, T.: 2000, A cryptographic solution to a game theoretic problem, *Advances in Cryptology — CRYPTO 2000, LNCS*, Vol. 1880, Springer-Verlag, pp. 112–130.
- Forges, F. M.: 1986, An approach to communication equilibria, *Econometrica* **54**(6), 1375–85.
- Forges, F. M.: 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 Myerson, R. B.: 2007, Sequential equilibria in bayesian games with communication, *Games and Economic Behavior* **60**(1), 104–134.
- Goldreich, O., Micali, S. and Wigderson, A.: 1987, How to play any mental game, *Proceedings of the 19th Symposium on Theory of Computing*, ACM, pp. 218–229.
- Goldwasser, S., Micali, S. and Rackoff, C.: 1985, The knowledge complexity of interactive proof-systems, *Proceedings of the 17th Symposium on Theory of Computing*, ACM, pp. 291–304. Final version in *SIAM Journal on Computing*, 1989, 186–208.
- Hopper, N., Langford, J. and von Ahn, L. a.: 2002, Provably secure steganography, *Proceedings of the Crypto 2006*.
- Izmalkov, S., Lepinski, M. and Micali, S.: 2005, Rational secure computation and ideal mechanism design, *Proceedings of the 46th Symposium on Foundations of Computer Science*, IEEE, pp. 585–594.
- Kocher, P., Jaffe, J., and June, B.: 1989, Differential Power Analysis, *Advances in Cryptology — CRYPTO 99*, vol. 1666 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 388–397.
- 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., Micali, S., Peikert, C. and Shelat, A.: 2004, Completely fair sfe and coalition-safe cheap talk, *Proceedings of the 23rd annual Symposium on Principles of distributed computing*, ACM, pp. 1–10.
- Myerson, R. B.: 1982, Optimal coordination mechanisms in generalized principal-agent problems, *Journal of Mathematical Economics* **10**(1), 67–81.
- Naor, M., Pinkas, B. and Sumner, R.: 1999, Privacy preserving auctions and mechanism design, *Proceedings of the 1st conference on Electronic Commerce*, ACM.
- Osborne, M. J. and Rubinstein, A.: 1997, *Game Theory*, MIT Press, Cambridge, Massachusetts.

- Rothkopf, M. H., Teisberg, T. J. and Kahn, E. P.: 1990, Why are Vickrey auctions rare?, *Journal of Political Economy* **98**, 94–109.
- Sjöström, T. and Maskin, E.: 2002, *Handbook of Social Choice and Welfare*, Vol. 1, North-Holland, chapter Implementation Theory, pp. 237–288.
- Urbano, A. and Vila, J. E.: 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 T. F. Bewley (ed.), *Advances in Economic Theory, Fifth World Congress*, Cambridge University Press, Cambridge, UK, pp. 33–70.
- Yao, A.: 1986, Protocol for secure two-party computation, never published. The result is presented in ?.