Verifiably Secure Devices

Sergei Izmalkov[1], Matt Lepinski[2], and Silvio Micali[3]

[1] MIT Department of Economics, izmalkov@mit.edu
[2] BBN Technologies, mlepinski@bbn.com
[3] MIT CSAIL, silvio@csail.mit.edu

Abstract. We put forward the notion of a *verifiably secure device*, in essence a stronger notion of secure computation, and achieve it in the ballot-box model. Verifiably secure devices

1. Provide a perfect solution to the problem of *achieving* correlated equilibrium, an important and extensively investigated problem at the intersection of game theory, cryptography and efficient algorithms; and
2. Enable the secure evaluation of multiple interdependent functions.

1 Introduction

From GMW to ILM1 Security. As put forward by Goldreich, Micali and Wigderson [11] (improving on two-party results of Yao [17]), secure computation consists of capturing **crucial aspects** of an abstract computation aided by a trusted party, by means of a concrete **implementation** that does not trust anyone. However, what is deemed crucial to capture and what constitutes an implementation have been changing over time. In order to achieve fundamental desiderata in a game theoretic setting, where incentives are added to the mixture and players are assumed to behave rationally, in [13] we put forward a stronger notion of secure computation, and achieved it in the *ballot-box model.* In essence, this is a physical model using ballots and a ballot randomizer, that is, the same "hardware" utilized from time immemorial for running a lottery and tallying secret votes. We refer to our 2005 notion as *ILM1 security.* Our main reason for introducing ILM1 security was implementing normal-form mechanisms in the stand-alone setting.

ILM2 Security. In this paper, we put forward a yet stronger notion of secure computation, herein referred to as *ILM2 security,* and achieve it in a variant of the ballot-box model. ILM2 security enables us to attain even more sophisticated applications. In particular, it enables us to (1) perfectly achieve *correlated equilibrium,* a crucial desideratum at the intersection of cryptography, game theory, and efficient algorithms; and (2) securely implement interdependent functions and mechanisms.

Setting Up A Comparison. In an fairer world, we could have come up with our present security notion in 2005. But the world is not fair, and we could not even conceive ILM2's security requirements back then. To ease their comparison, we sketch both approaches in the next two subsections, initially focussing on the secure evaluation of a single, probabilistic, finite function f from n inputs to $n+1$ outputs. Without loss of generality, $f : (\{0,1\}^a)^n \to (\{0,1\}^b)^{n+1}$.

We break each approach into the following components: (1) *The Ideal Evaluation*, describing the "target computation", that is, an evaluation of f with the help of a trusted party; (2) *The Concrete Model*, highlighting the "mechanics" of an evaluation of f without a trusted party; and (3) *The Security Notion*, describing the extent to which the concrete model captures the ideal evaluation.

Traditionally, in summarizing secure computation, the second and third components are merged. In the original GMW definition, there was certainly no compelling need to treat the concrete model as a "variable." Subsequently, their concrete model (i.e., a communication protocol among all players) persisted in all variants of secure computation, thus possibly generating a sense of "inevitability." We instead highlight the concrete model as an independent component because one of our contributions is indeed a change of scenery of this aspect of secure computation.

1.1 The ILM1 Approach

The Ideal Evaluation. In the ILM1 approach, an ideal evaluation of f proceeds in three stages.

1. In the input stage, each player i either (1.1) publicly aborts, in which case the entire evaluation ends, or (1.2) privately and independently chooses an a-bit string x_i and privately sends it to T.
2. In the computation stage, T publicizes a random string σ, and then privately evaluates f on all x_i's so as to obtain the b-bit values $y, y_1, \ldots, y_n$.
3. In the output stage, T publicizes y and privately hands back y_i to each player i.

(Note that the players do not communicate in an ILM1 ideal evaluation. By contrast, in the GMW approach, the ideal evaluation of f cannot be precisely matched by a GMW-secure protocol unless it offers the players the option of communicating to each other prior to aborting or sending T their inputs.)

The Concrete Model. In the ILM1 approach, the concrete model for mimicking an ideal evaluation of f continues to be that of the original GMW approach: namely, a multi-round communication protocol P_f executed by all players, where each player secretly holds and updates a share of the global state of the evaluation. The only difference is in the communication model. That is, rather than relying on broadcasting and/or private channels (as traditional GMW-secure protocols do), ILM1 protocols rely on ballots and a ballot randomizer.

The Security Notion. At the highest level, for P_f to be an ILM1-secure protocol for a function f, there must be an "output-preserving" bijection between the players'non-aborting strategies in P_f and the players'non-aborting strategies in an ideal evaluation of f.[4]

Furthermore, to match the privacy of f's ideal evaluation, it is required that

1. During the input-commitment stage the only information collectively available to any set of the players about the inputs of the other players consists of a fixed string —which is no information at all, and strictly less than observing a random string;[5]
2. During the computation stage the only information collectively available to any set of the players about the inputs of the other players consists of a common random string; and
3. During the output stage, the only information collectively available to any set of the players about the inputs of the other players consists of the desired outcome —in a pre-specified, deterministic encoding.

(Note that our use of the term "collectively available information" in relation to a set of the players does not refer to information known to at least a member of the set. Rather, it refers to the information that one would obtain were he able to join together the information available to each member of the set. Barring the existence of external and undetected means of communication, an individual member of the set can only get such "collective" information by querying the other members after the protocol P_f terminates. As deducible by the above sketchy description, an ILM1-secure protocol does not enable any inter-player communication, as demanded in the ideal evaluation.)

Main Properties of ILM1 Security

- *Perfect implementation of Normal-Form Mechanisms in the Stand-Alone Setting.* Our original reason for introducing the ILM1 notion of security was to be able to perfectly implement normal form mechanisms in the stand-alone setting, something that was not implied by GMW-secure computation. We refer the reader to [13] for the definition of such a perfect implementation (and actually to MIT-CSAIL-TR-2007-040 for a more precise explanation). Here we are happy to quickly recall what a normal-form mechanism and the stand-alone setting are.

[4] By contrast, a player has "much more to do" in a GMW-secure protocol P_f than in an ideal evaluation of f. In the latter setting, in fact, there are exactly $2^a + 1$ "strategies" for a player i, one for each possible i-input to f plus aborting. Accordinly, if f operates on 10-bit inputs, the total number of strategies is roughly one thousand. By contrast, letting for concreteness P_f be the original protocol of [11], player i not only has to decide between 2^a inputs or aborting, but can also decide which encryptions to broadcast, which query bits to use in a zero-knowledge proof and so on. Thus, while each player has roughly 1000 strategies in an ideal evaluation of h, he may easily have more than 2^{1000} strategies in P_f. Such richness of strategies severely limits the relevance of GMW-secure protocols to game-theoretic applications.

[5] Quite differently, in a GMW-secure protocol for f, the players —by broadcasting and/or privately exchanging all kinds of strings— can make available plenty of additional information.

Very informally, a (finite) normal-form mechanism is a function $f : (\{0,1\}^a)^n \rightarrow (\{0,1\}^a)^{n+1}$. The mechanism play coincides with an ideal evaluation of f as in the ILM1 approach. However, such play is analyzed as a game in a context specifying players' preferences and thus the players get different utilities (think of "dollars prizes") for different outputs of f.

Very informally too, by a normal-form mechanism in the stand-alone setting we mean that "nothing else happens after a single play of the mechanism."

- *Fairness and Perfect Security without Honest Majority.* Informally fairness means that either all (honest) players receive their right outputs, or nobody does. It is a notable property of ILM1 security that it simultaneously guarantees fairness and perfect information-theoretic security without relying on the majority of the players to be honest. Here by a "honest" player we mean one sticking to his protocol instructions no matter what.
(GMW-secure protocols —relying on broadcasting and/or private channels— do not guarantee fairness unless the majority of the players are honest. Indeed, Cleve [7] shows that not even the the probabilistic function that, on no input, outputs a random bit can be fairly and efficiently computed when half of the players can abort the protocol —let alone maliciously deviate from their instructions. Remarkably, in 2004, Lepinski, Micali, Peikert and Shelat [15] put forward a protocol guaranteeing fairness without any honest majority in a mixed model involving broadcasting and regular ballots.[6] The security of their protocol, however, was only computational.)
- *Perfect Security and Universal Composibility Without Honest Majority.* ILM1-secure protocols satisfy "composibility" as defined in 2000 by Dodis and Micali [9]. It is by now recognized that their notion actually coincides, in the perfect information-theoretic setting, with universal composibility as defined in 2001 by Canetti [6]. Indeed, Halevi and Rabin show that *perfect* simulatability via *straight-line* simulators (as demanded in [9]) implies universal composibility.

  The universal composibility of ILM1-secure protocols is remarkable because it is achieved together with perfect information-theoretic security and without relying on any honest majority, something that was not known to be possible in the past.

1.2 The ILM2 Approach

In the ILM2 approach, an ideal evaluation of f continues to proceed in three stages. There are two possibilities for the first stage: one with aborts and one without. We refer to the first one as *weak* ideal evaluation, and the second one as *strong* ideal evaluation. With start by presenting the latter, simpler and more provocative notion.

[6] I.e., their ballots needed not to be identical, nor was a ballot randomizer needed for their construction.

Strong Ideal Evaluation.

1. In the input stage, each player i —independently from all others— secretly chooses an input x_i and gives it to T in an envelope,
2. In the computation stage, T privately opens all received envelopes and then privately computes $y, y_1, \ldots, y_n = f(x_1, \ldots, x_n)$.
3. In the output stage, T publicizes y and publicly hands back to each player i an envelope containing y_i.

The Concrete Model. In the ILM2 approach, the concrete model continues to rely on the same hardware of the ILM1 approach (i.e., identical ballots and a ballot randomizer), but no longer coincides with a communication protocol among the players. Instead, the encoding of the global state is now fully contained in a sequence of envelopes publicly manipulated by a *verifiable* entity T'. Let us explain.

In the ILM2 ideal evaluation of a function f, T is *trusted.* Indeed, he could change the outputs and/or reveal undue information about the players' inputs (e.g., via the envelopes he hands back to the players or after the evaluation) without any fear of being "caught." By contrast, to securely evaluate f, T' is called to perform a *unique sequence of ballot operations* such that the players *can verify that the right operations have been performed.* For instance, if privacy did not matter, upon receiving the envelopes containing the players' inputs, T' could be required to open all of them. In which case it would be trivial to check whether he has done what was required of him. (To be sure, such a verifiable T' would still be trusted not to —say— publicly open half of the envelopes and destroy the other half. But such trust is much milder: because any deviation from opening all envelopes would become of public record, T' can be kept accountable.)

Because the actions required from T' are uniquely determined and verifiable, human or not, we think of T' as a *verifiable device.*

The Security Notion. The ILM2 security notion is the most stringent we can (currently) conceive. Before sketching it, it should be realized that, in whatever model used (encrypt-and-broadcast, private-channel, ballot-box, etc.) each "action" essentially has a public and a private informational component.[7] For instance, in the ballot-box model, the action of "publicly opening envelope j" generates only a public component: namely, "(publiclyopen,j,c)" where c is the now exposed content of (former) envelope j. As for another example, "party i privately opens envelope j" generates a public component "(privatelyopen, i,j)" together with the private component c for party i, where c is the content of the (former) envelope j. The correctness requirements of an ILM2 concrete evaluation of f are not surprising (and are formally presented later on). The privacy requirements are instead surprising. Namely, in a verified computation,

[7] In the encrypt-and-broadcast model, when player i sends a message m to player j encrypted with j's key, the public component is "ciphertext C from i to j" while the private component to j is "m".

1. In a correct execution of the verifiable device, the public history generated (i.e., the concatenation of all public informational components) is a *fixed* string R (depending only on f) and *vice versa* whenever the public history of an execution of the verifiable device is R, then f has been evaluated correctly and privately on the actual inputs, no matter what they may be;
2. The private history of each player i consists only of his own input x_i; and
3. The private history of the verifiable device consists only of a random string.

Remarks.

- *Perfect Security and Universal Composibility without Honest Majority.* As for the ILM1 case, this is due to the fact that ILM2 security satisfies the Dodis-Micali conditions.
- *Perfect Determinism, and Perfect Independence.* ILM2 security is the only notion of secure computation that is totally deterministic to the players. They do not observe any common randomness and do not even generate any local randomness. The situation is not too different for the verifiable device. Namely, he does not generate any local randomness, but individually observes a random string ρ. Such string however cannot be communicated by the device to the players by means of the operations available to him (which are verifiable anyway). And should the device reveal ρ to some players afterwards, it would be "too late." During the executions the players have been *absolutely isolated from one another.*
- *Hidden Aborts.* Let us now explain in what sense it is meaningful to consider ideal executions in which players cannot abort. In a typical secure protocol, it makes no sense to worry about player i learning j's input by pointing a gun at j's head. The analysis of any protocol should be relative only to the actions available within the protocol's model. Nonetheless, aborting is quite possible within the confines of a protocol's actions. For instance, a player who is required to broadcast the decryption of a given cipher-text might be able to cause an abort by broadcasting a different string. That is, one ability of aborting arises when the set of available actions is richer than that "handleable" by the protocol. This source of aborts, however, may not be always present, and is not be present in the ILM2 case. Nonetheless, there is one more source of aborts: namely, taking "no action." That is, aborts may also occur in models for which "doing nothing" is distinguishable from "the prescribed actions". In the broadcast model, if a player is envisaged to broadcast a bit, broadcasting nothing is quite a different thing. Doing nothing is also distinguishable from all possible actions in the version of the ballot-box model envisaged in ILM1-secure protocols, and easily enables a player to halt the joint computation. Indeed, in the ILM1 approach, the global state of the concrete computation is shared in n pieces, each known to a different player, and each necessary for the progress of the computation. Thus, if a player suicides carrying his own piece to the other world, the computation cannot be continued in any meaningful way.

By contrast, in an ILM2-secure protocol, after the verifiable device starts taking public actions on the envelopes encoding the players' inputs, all envelopes are in the hands of the device, and the global state of the computation is contained fully in those envelopes. Thus, from that point on, in a properly verified execution the computation goes on (correctly and privately) independently of what the players may or may not do. Device abort is not an issue either. Because our devices are called to do a single verifiable action at every point of their executions, only device no-action is meaningful to analyze, and it is a notable property of our construction that, if needed, the verifiable device can be substituted at any moment with another verifiable device without any loss.

In an ILM2-secure protocol, therefore, the only stage in which the issue of abort might possibly arise is the input stage; because there the players' participation is crucial to ensure that envelopes properly encoding their inputs are handed to the device. There, however, we have engineered players' input commitment so as to "hide aborts." Conceptually, a player contributes an input bit to the evaluation of f as follows. First, he publicly receives from the device two envelopes, both publicly generated with different prescribed contents. Then the player is asked to secretly permute them: leaving them in the same order corresponds to secretly inputting 0, flipping their order corresponds to secretly inputting 1. Formally speaking, therefore, aborting is indistinguishable from secretly inputting 0.

- *Public Aborts.* Practically speaking, however, enforcing "abort indistinguishability" requires building and analyzing some ad hoc simple gadget with an associated protocol. (If you fail in designing them, just ask us!) Such a gadget, of course, would be a physical assumption not only *additional* to ballots and ballot-boxes, but also *quite new,* while ballots and ballot-randomizers have been around for time immemorial and are thus easy to accept as "physical axioms."[8] Altogether, most readers would prefer to stick with the more intuitive operation of "player i secretly permutes two envelopes, or publicly aborts." In this case, we can only achieve the following type of ideal evaluation.

  WEAK IDEAL EVALUATION.

  1′. In the input stage, each player i —independently from all others— either secretly chooses an input x_i and gives it to T in an envelope, or publicly gives T the input $x_i = 0^a$ —i.e., the concatenation of a 0s.
  2. In the computation stage, T privately opens all received envelopes and then privately computes $y, y_1, \ldots, y_n = f(x_1, \ldots, x_n)$.
  3. In the output stage, T publicizes y and publicly hands back to each player i an envelope containing y_i.

[8] Put it this way: if you think that it is not possible to randomize identical ballots, you should also explain (1) why people have been shuffling decks of cards for ever and for real money; *and* (2) why electoral precincts do not simplify voting procedures by always adopting roll-call voting.

Definitionally, it is clear that that the strong and weak versions of ILM2 security are both stronger than ILM1 security. Let us now present two specific concrete settings requiring ILM2 security. The first one actually involves the implementation of function with no inputs. Therefore it is does not matter whether the players can or cannot abort! Thus, weak or strong, ILM2 security wins anyway.

1.3 Perfect Achievement of Correlated Equilibrium

Consider the following trivial probabilistic function f that, on no inputs, outputs a pair of strings:

$f() = (C, G)$, (C, G), *or* (D, G) *—each with probability* $1/3$.

Obviously, using proper binary encodings, an ILM1-secure protocol P_f for f exists. Ideally, such a P_f should successfully replace an ideal evaluation for f in any setting. However, *this is not true for the following setting.*

	E	F	G	H
A	$100, 0$	$-\infty, -\infty$	$-\infty, -\infty$	$-\infty, -\infty$
B	$-\infty, -\infty$	$0, 100$	$-\infty, -\infty$	$-\infty, -\infty$
C	$-\infty, -\infty$	$-\infty, -\infty$	$4, 4$	$1, 5$
D	$-\infty, -\infty$	$-\infty, -\infty$	$5, 1$	$0, 0$

Let Row and Column be the two players of the normal-form game $\mathbb{G}$ described by the above pay-off matrix. Accordingly, Row and Column are confined to two separate rooms: the first facing a keyboard with 4 buttons A,B,C and D; and the second with a keyboard whose buttons are E,F,G, and H. Without any external help, only the following (reasonable) Nash equilibria exist in $\mathbb{G}$: (A,E), which is very good for Row, (B,F), which is very good for Column, (C, H), (D, G), and the mixed equilibrium $(\frac{1}{2}C + \frac{1}{2}D, \frac{1}{2}G + \frac{1}{2}H)$, which yields payoff 2.5 to each player. Mankind's interest, however, is that the outcome of $\mathbb{G}$ is either (C, G), (C, H), or (D, G), each with probability $\frac{1}{3}$. Accordingly, an angel in all his splendor descends from the sky, privately evaluates the above function $f()$ to obtain a pair (X, Y), tells Row and Column that he has done so, and then provides Row with an envelope containing X, and Column with an envelope containing Y. Technically, this angelic intervention puts Row and Column in a *correlated equilibrium* $\mathbb{A}$, for "angelic". (Correlated equilibria were proposed by Aumann [1].) In essence, each player is better off playing the recommendation received from the angel if he believes the other will do so. It is evident that the expected utility of each player in $\mathbb{A}$ is $10/3$. Clearly, however, Row and Column would prefer to play a different correlated equilibrium: namely $\mathbb{E}$, the correlated equilibrium corresponding to selecting (A,E) or (B,F), each with probability $1/2$. Unfortunately, they cannot reach such equilibrium without external help. Nor can they use the X and Y they respectively received by the angel (which is external help indeed!) in order to "translate" their angelic recommendations into their preferable ones. The point is that if X=C, then Row has no idea whether Y=G or Y=H. Each is equally probable to him. (Symmetrically, if Y=G, then Column has no idea whether X=C or X=D, both are equally probable to him.)

If X=D, then Row knows that Y=G, but he also know that Column has no idea whether X=C or X=D. Accordingly, the expected payoff provided to the players by any "translation strategy" is $-\infty$. This must be very frustrating, since (1) Row and Column receive an expected utility of 50 each in $\mathbb{E}$, and (2) Row and Column only need a *single* random bit to coordinate their actions to achieve $\mathbb{E}$!

Consider now replacing the angel for f with the ILM1-secure protocol P_f. (This is indeed possible because ILM1 security does not require any honest majority.) Then, after correctly and privately computing their P_f-recommendations, Row and Column can simply ignore them and use instead the first common random bit generated by P_f to coordinate and play equilibrium $\mathbb{E}$ instead. (Indeed, at some point of any execution of P_f, 5 envelopes previously randomized by the ballot-box are publicly opened, thus publicly revealing a random permutation of 5 elements, which can be trivially translated into a single random bit.)

How is this possible in view of the claim that ILM1 successfully evaluates any finite function in the stand-alone setting? The answer is that *the above setting is not stand alone.* Indeed, Row and Column are not just asked to play P_f and get immediately rewarded. They are asked to play first P_f and *then* $\mathbb{G}$, and they ultimately get $\mathbb{G}$'s rewards. Thus, even a minimal deviation from the stand-alone model, together with the minimal presence of common random bit, sets apart what can happen with the help of an angel and what can happen with traditional secure computation. In other words, so far in secure computation we have been conditioned to think (and the last author agrees to take some blame) that "randomness and efficient computation are for free." Unfortunately, this is not true in game-theoretic settings. In general,

> Common randomness *as a side product of secure computation is akin to* pollution *as a side product of energy transformation.*

A Perfect Solution to a Beautiful Problem. As introduced by Aumann [1], a correlated equilibrium for a normal-form game G is a probability distribution f over the actions available to the players such that if —somehow!— a profile of "recommended actions" $(a_1, \ldots, a_n)$ is chosen according to f, and each player i learns a_i without gaining any other information about a_{-i}, then no single player can deviate from his recommendation in a play of G and improve his expected utility. Given the description of any equilibrium E, the obvious problem consists of finding a concrete way that precisely simulates a trusted party correctly sampling and privately handing out to the players E's recommendations.

Despite much work [2, 5, 10, 8, 16, 15, 12], all prior solutions to this problem were imperfect. Specific deficiencies included limited rationality, strategic-option alteration, limitations on the numbers of players, infeasible computation and resources, and imposing a pre-specified beliefs to yet-to-be-determined players.

By contrast ILM2 security provides a perfect solution to the achieving correlated equilibrium. (Details will be given in the final paper.) Note that the main concerns here is orthogonal to composibility, that does not care about preserving strategic opportunities. Indeed, generating a common random bit is OK vis à vis composibility, but alters the achievement of correlated equilibrium.

1.4 Interdependent Secure Computations and Mechanisms

More generally, we now consider evaluating multiple *interdependent* functions, each not only receiving fresh inputs from the players and producing public and private outputs, but also receiving an additional secret input consisting of state information from the evaluation of another function, and passing some state information of its own evaluation as an additional secret input to another function. For simplicity sake, below we formalize just the case of a pre-defined linear sequence of such functions. (In particular, we could handle trusted parties can operate more than once, each time passing their state to different trusted parties, operate simultaneously, recursively, et cetera. Of course, the more complex the setting, the richer are the strategic opportunities of the players. We are not interested in analyzing them, but rather to match them exactly, whatever they may be —without envisaging concurrently executing extraneous secure protocols.)

(Weak) Ideal Evaluation of Interdependent Functions. Let F be a sequence of functions, $F = f_1, \ldots, f_k$, where $f_i : (\{0,1\}^a)^{n+1} \rightarrow (\{0,1\}^a)^{n+2}$. Letting s_0 and s_{k+1} be empty strings, an ideal evaluation of F proceeds in k phases, with the help of k separate trusted parties: $T_1, \ldots, T_k$. The jth phase consists of 3 stages.

1. In the input stage, T_j secretly receives state information s_{j-1} and publicly receives the identities of all aborting players. Further, each non-aborting player i independently and secretly chooses an input x_i^j and gives it to T in an envelope, or publicly aborts. Each aborting player i publicly gives T_j the input $x_i^j = 0^a$ —i.e., the concatenation of a 0s.
2. In the computation stage, T_j privately opens all received envelopes and then privately computes
$$s_j, y^j, y_1^j, \ldots, y_n^j = f_j(s_{j-1}, x_1^j, \ldots, x_n^j).$$
3. In the output stage, T_j publicizes y^j, privately hands to each player i an envelope containing y_i^j, and privately hands s_j to T_{j+1}.

Note that this weak evaluation can be changed in several ways if desired or necessary to model the situation at hand. For instance, rather than forcing aborting players to always contribute the input 0^a in the future, one may envisage a player aborting in the input stage of phase j as contributing the input 0^a in just that phase, but give him an opportunity to fully participate in future phases. This may be a natural choice, but of course it enlarges the players' signalling abilities. As for another possible change, one may demand that no envelope containing the private output of an aborting player be ever given to him. A third alternative may consist of banning the aborting players from future participation, thus changing the functions of future phases so as to have fewer inputs (although this technically is already beyond the simpler setting envisaged above). And so on. It all depends on the setting we wish to model.

The "strong" version of the above definition can be obtained by removing the possibility of aborting altogether.

To go from sequences of functions to the analysis of sequences of normal-form mechanisms one needs only to specify the players' preferences over the outcomes.

The Inachievability of Secure Interdependency with Prior Notions. ILM1 security does not suffice for meaningfully implementing a sequence of inter-dependent functions/mechanisms. (GMW security would be even less meaningful, particularly if relying on private channels.) The main problem is as follows. Assume that player i aborts in the jth phase. In the ideal setting, i necessarily aborts in the input stage. Accordingly, T_j uses 0^a as i's input for the function f_j, completes the computation of f_j, returns all public and private outputs to the right players, and finally provides T_{j+1} with the correct state information s_j. This is the case because in the ideal setting T_j has already received from T_{j-1} the state information s_{j-1}. By contrast, when i aborts the execution of an ILM1-secure protocol P_{f_j} for f_j, he makes his own share of the global computation disappear with him, causing the entire sequence of evaluations to grind to a halt. (In fact, any way of continuing would proceed with an incorrect state: the other players do have their own shares of the current global state, but their shares alone are consistent with any feasible global state at that point.) If the whole evaluation were just a mental game, endowing a player with the ability of halting the entire sequence of future evaluations by his aborting in specific phase might not matter. But causing the entire sequence of evaluations to abort may be disastrous in other settings, where "real incentives" are associated to the whole enterprize. For instance, assume that the government of a major country is privatizing its national resources (it has happened before!) by means of a complex sequence of inter-dependent normal-form mechanisms, so as to achieve complex social objectives. Gas has been allocated first, oil second, and so on, with the players paying real money for these resources (and possibly selling off assets they previously owned in order to raise the necessary cash). And then, suddenly, one of players commits suicide. What should the government do? Sending every one home, as if nothing ever happened, and demanding that the allocated resources be returned is not an option: who is going to return the assets some players had to sell (possibly in different countries) in order to win some of the present resources? Nor is it an option to recognize all allocations already made and stop the future ones. In fact, due to the interdependency of the mechanisms in the sequence, a player may have chosen to acquire a given resource in one of the early phases (by strategically choosing his secret inputs to the first functions in the sequence) only in order to improve his chance to win resources more attractive to him in the future phases. Nor is it an option to allocate the remaining resources by means of a different sequence of mechanisms. The players' past strategies depended on that very evaluation to continue with the right state.[9]

[9] Notice that although exogenous incentives, such as fines, may discourage abortions, they are incapable of perfectly solving the problem. On one hand, fining players will not resurrect the right computation state. On the other, finding the right fine to impose is not an easy theoretical problem. Assume that a player i aborts because, based on his received private information, he realizes that the rest of the mechanisms —if continued— would cause him immense financial harm. Then, to induce him not to do so, a mechanism designer must impose a fine greater than i's expected loss. But, in general, the designer may be unaware of the players' preferences.

In essence, not every thing in life is a mental game without incentives, and to properly deal with these incentives one needs to preserve the originally envisaged strategic opportunities. It should be clear that weak ILM2 security is instead capable of matching the above ideal scenario. Indeed, in ILM2-secure computation, the global state continues to remain available to each verifiable device D_j (corresponding to trusted party T_j) whether the players abort or not. Moreover, the players do not have the ability to signal in any additional way from one mechanism to the next. Not even random strings will enable to alter the strategic opportunities available to the players in a weak-ILM2-secure protocol.

Finally, note that one may also consider *strong* ideal evaluations of interdependent functions, and that strong-ILM2-secure protocols will be able to match these more stringent requirements.

2 Notation

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

If x is a sequence, by either x^i or x_i we denote x's ith element,[10] and by $\{x\}$ the set $\{z : x^i = z \text{ for some } i\}$. If x is a sequence of k integers, and m is an integer, by $x + m$ we denote the sequence $x^1 + m, \ldots, x^k + m$. If x and y are sequences, respectively of length j and k, by $x \circ y$ we denote their concatenation (i.e., the sequence of $j + k$ elements whose ith element is x^i if $i \leq j$, and y^{i-j} otherwise). If x and y are strings (i.e., sequences with elements in Σ), we denote their concatenation by xy.

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

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

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

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

3 The Ballot-Box Model

The ballot-box model ultimately is an abstract model of communication, but possesses a quite natural physical interpretation. The physical setting is that of a group of players, 1 through n, and a distinguished "player" 0, the device, seated around a table together and 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. Indeed, in this paper, we envisage more actions than in [13].

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 10 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, and its consequences are the same no matter who performs it. These 7 classes are: (1) publicly write a symbol on a piece of paper and seal it into a new, empty envelope; (2) publicly open an envelope to reveal its content to all players; (3) publicly seal a sequence of envelopes into a new super-envelope; (4) publicly open a super-envelope to expose its inner envelopes; (5) publicly reorder a sequence of envelopes; (6) publicly destroy a ballot; and (7) do nothing. The last three classes just simplify the description of our construction.

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, reordering the chosen ballots according to a permutation randomly chosen by —and solely known to— Nature.

Each action of 9th and 10th classes is referred to as a *private action*, because some details about either its inputs or outputs are known solely to the player (or device) performing it. These two classes are: (9) privately open, read the content, and reseal an envelope; and (10) secretly reorder a sequence of envelopes. We imagine that the players observe what ballots these actions are performed upon, but the actions themselves are performed outside of public view. For instance, to perform an action of class 10, a player can shuffle the envelopes behind his back or within a box, so that only he knows in what order the envelopes are returned on the table.

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, i, j, k, l, ..."; where A is a string identifying the action, i is the number corresponding to the player performing the action, and $j, k, l, ...$ are the identifiers of the ballots involved. If the action is public, for convenience, the identity of the player performing it is not recorded, since the effect of the action is the same no matter by whom the action is performed. The *public history* is the concatenation of the public strings generated by all actions executed thus far. Similarly, the *private history* of each player is the concatenation of the private strings generated by the private actions performed by, respectively, the player. The private string is the content of the opened envelope for a "private read" action and the actual permutation for a "secret permute" action.

3.2 Formalization

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 $cont_B(j)$ denotes the content of j; if $x = j^1, \ldots, j^k$ is a sequence of envelope identifiers in I_B, then $cont_B(x)$ denotes the concatenation of the contents of these envelopes, that is, the string $cont_B(j^1) \cdots cont_B(j^k)$.

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

- B is a well defined set of ballots;
- R is a sequence of strings in Σ^*, $R = R^1, R^2, \ldots$; and
- H a tuple of sequences of strings in Σ^*, $H = H_0, H_1, \ldots, H_n$.

We refer to B as the *ballot set;* to R as the *public history;* to each element of R as a *record;* to H as the *private history*; to H_0 as the *private history of the device*; and to each H_i as the *private history of player* i. The *empty global memory* is the global memory for which the ballot set, the public history, and all the private histories are all 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 10 classes. For each $a \in \mathcal{A}_{gm}$ we provide a formal identifier; an informal reference (to facilitate the high-level description of our constructions); and a functional specification. If $gm = (B, R, H)$, we actually specify $a(gm)$ as a program acting on variables B, R, and H. For convenience, we include in R the auxiliary variable ub, the *identifier upper-bound*: a value equal to 0 for an empty global memory, and always greater than or equal to any identifier in I_B.

1. $(\textsc{NewEn}, c)$ —where $c \in \Sigma$.
   "Make a new envelope with public content c."
   $\mathtt{ub} := \mathtt{ub} + 1$; $B := B \cup \{(\mathtt{ub}, c, 0)\}$; and $R := R \circ (\textsc{NewEn}, c, \mathtt{ub})$.
2. $(\textsc{OpenEn}, j)$ —where j is an envelope identifier in I_B.
   "Publicly open envelope j to reveal content $cont_B(j)$."
   $B := B \setminus \{B_j\}$ and $R := R \circ (\textsc{OpenEn}, j, cont_B(j), \mathtt{ub})$.
3. $(\textsc{NewSup}, j_1, \ldots, j_L)$ —where $L \leq 5$, and $j_1, \ldots, j_L \in I_B$ are distinct envelope identifiers.
   "Make a new super-envelope containing the envelopes $j_1, \ldots, j_L$."
   $\mathtt{ub} := \mathtt{ub} + 1$; $B := B \cup \{(\mathtt{ub}, (cont_B(j_1), \ldots, (cont_B(j_L)), L)\}$;
   $B := B \setminus \{B_{j_1}, \ldots, B_{j_L}\}$; and $R := R \circ (\textsc{NewSup}, j_1, \ldots, j_L, \mathtt{ub})$.
4. $(\textsc{OpenSup}, j)$ —where $j \in I_B$ is the identifier of a level-L super-envelope.[11]
   "Open super-envelope j."
   letting $cont_B(j) = (c_1, \ldots, c_L)$, $B := B \cup \{(\mathtt{ub}+1, c_1, 0), \ldots, (\mathtt{ub}+L, c_L, 0)\}$;
   $B := B \setminus \{B_j\}$; $\mathtt{ub} := \mathtt{ub} + L$; and $R := R \circ (\textsc{OpenSup}, j, \mathtt{ub})$.
5. $(\textsc{PublicPermute}, j_1, \ldots, j_k, p)$ —where $k \leq 5$, $j_1, \ldots j_k \in I_B$ are distinct identifiers of ballots of the same level L, and $p \in \mathrm{SYM}_k$.
   "Publicly permute $j_1, \ldots, j_k$ according to p."
   $B := B \cup \{(\mathtt{ub}+1, cont_B(j_{p(1)}), L), \ldots, (\mathtt{ub}+k, cont_B(j_{p(K)}), L)\}$; $\mathtt{ub} := \mathtt{ub} + k$; $B := B \setminus \{B_{j_1}, \ldots, B_{j_k}\}$; and $R := R \circ (\textsc{PublicPermute}, j_1, \ldots, j_k, p, \mathtt{ub})$.
6. $(\textsc{Destroy}, j)$ —where j is a ballot identifier in I_B.
   "Destroy ballot j"
   $B := B \setminus \{B_j\}$ and $R := R \circ (\textsc{Destroy}, j, \mathtt{ub})$.
7. $(\textsc{DoNothing})$.
   "Do nothing"
   $B := B$ and $R := R \circ (\textsc{DoNothing}, \mathtt{ub})$.
8. $(\textsc{BallotBox}, j_1, \ldots, j_k)$ — where $k \leq 5$ and $j_1, \ldots j_k \in I_B$ are distinct identifiers of ballots of the same level L.
   "Ballotbox $j_1, \ldots, j_k$"
   $p \leftarrow rand(\mathrm{SYM}_k)$; $B := B \cup \{(\mathtt{ub}+p(1), cont_B(j_1), L), \ldots, (\mathtt{ub}+p(k), cont_B(j_k), L)\}$;
   $B := B \setminus \{B_{j_1}, \ldots, B_{j_k}\}$; $\mathtt{ub} := \mathtt{ub}+k$; and $R := R \circ (\textsc{BallotBox}, j_1, \ldots, j_k, \mathtt{ub})$.
9. $(\textsc{PrivRead}, i, j)$ —where $i \in [0, n]$ and j is an envelope identifier in I_B.
   "i privately reads and reseals envelope j."
   $R := R \circ (\textsc{PrivRead}, i, j, \mathtt{ub})$ and $H_i := H_i \circ cont_B(j)$.

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

10. $(\textsc{SecretPermute}, i, j_1, \ldots, j_k, p)$ —where $i \in [0, n]$, $k \leq 5$, $p \in \mathrm{SYM}_k$, and $j_1, \ldots j_k \in I_B$ are distinct identifiers of ballots with the same level L.
"i secretly permutes $j_1, \ldots, j_k$ (according to p)."
$B := B \cup \{(\mathtt{ub}+1, cont_B(j_{p(1)}), L), \ldots, (\mathtt{ub}+k, cont_B(j_{p(K)}), L)\}$; $B := B \setminus \{B_{j_1}, \ldots, B_{j_k}\}$; $\mathtt{ub} := \mathtt{ub}+k$; $R := R \circ (\textsc{SecretPermute}, i, j_1 \ldots, j_k, \mathtt{ub})$, and $H_i := H_i \circ p$.

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, BallotBox, and SecretPermute 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 $gm^0, gm^1, \ldots, gm^k$, such that gm^0 is the empty global memory; $gm^k = gm$; and, for all $i \in [1, k]$, $gm^i = a^i(gm^{i-1})$ for some $a^i \in \mathcal{A}_{gm^{i-1}}$.*

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

Remark. If $gm = (B, R, H)$ is feasible, then $\mathcal{A}_{gm}$ is easily computable from R alone (and so is `ub`). 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 history also have the same set of available actions. This motivates the following definition.

Definition 2. *If R is a feasible public history, by $\mathcal{A}_R$ we denote the set of available actions for any feasible global memory with public history R.*

4 The Notion of a (Not Necessarily Verifiable) Device

Definition 3. *Let $\mathcal{D}$ be a sequence of K functions. We say that $\mathcal{D}$ is a* ballot-box device (of length K) *if, for all $k \in [1, K]$, public histories R and private histories H_0, $\mathcal{D}^k(R, H_0)$ specifies a single action. If a private action is specified, then it has $i = 0$.*

An execution of $\mathcal{D}$ *on an initial feasible global memory (B^0, R^0, H^0) is a sequence of global memories $(B^0, R^0, H^0), \ldots, (B^K, R^K, H^K)$ such that $(B^k, R^k, H^k) = a^k(B^{k-1}, R^{k-1}, H^{k-1})$ for all $k \in [1, K]$, where $a^k = \mathcal{D}^k(R^{k-1}, H_0^{k-1})$.*

If e is an execution of $\mathcal{D}$*, by* $B^k(e)$*,* $R^k(e)$*, and* $H^k(e)$ *we denote, respectively, the ballot set, the public history, and the private history of e at round k.*

By $R^k_{\mathcal{D}}(e)$ *and* $H^k_{0,\mathcal{D}}(e)$ *we denote, respectively, the last k records of* $R^k(e)$ *and* $H^k_0(e) \setminus H^0_0$ *(i.e., "the records appended to* R^0 *and* H^0_0 *by executing* $\mathcal{D}$*").*

The executions of $\mathcal{D}$ on initial memory gm^0 constitute a distribution,[12] which we denote by $EX_{\mathcal{D}}(gm^0)$.

Remarks.

- Note that if $\mathcal{D} = \mathcal{D}^1, \ldots, \mathcal{D}^K$ and $\mathcal{T} = \mathcal{T}^1, \ldots, \mathcal{T}^L$ are ballot-box devices, then their concatenation, that is, $\mathcal{D}^1, \ldots, \mathcal{D}^K, \mathcal{T}^1, \ldots, \mathcal{T}^L$ is a ballot-box device too.

5 The Notion of a Verifiably Secure Computer

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

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

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

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

With modularity in mind, we actually envision that an execution of a computer $\mathcal{D}$ may be preceded and/or followed by the execution of other computers. We thus insist that $\mathcal{D}$ does not "touch" any ballots of the initial memory besides its input envelopes. This way, partial results already computed, if any, will remain intact.

Definition 6. *Let* $f : X^a \to Y^b$ *be a finite function, where* $X, Y \subset \Sigma^*$*; let x and y be two address vectors. We say that a ballot-box device* $\mathcal{D}$ *is a* verifiably secure computer for *f, with input address vector x and output address vector y, if there exist a constant sequence U and a* straight-line *no-input simulator SIM such*

[12] Indeed, although each function $\mathcal{D}^k$ is deterministic, $\mathcal{D}^k(R)$ may return an action of nature.

that, for any execution e of $\mathcal{D}$ on an initial memory $gm^0 = (B^0, R^0, H^0)$, proper for f and x and with identifier upper-bound $\mathtt{ub}_0$, the following three properties hold:

1. Correctness: $cont_{B^K(e)}(y + \mathtt{ub}) = f(cont_{B^0}(x))$.
2. Privacy: $R^K_{\mathcal{D}}(e) = U$ *and* $H^K_{0,\mathcal{D}}(e) = SIM()$.
3. Clean Operation: $B^K(e) = B_{\{y+\mathtt{ub}\}} \cup B^0 \setminus B_{\{x\}}$.

We refer to SIM as $\mathcal{D}$'s simulator*; to $B_{\{x\}}$ as the* input envelopes*; and to $B_{\{y+\mathtt{ub}\}}$ as the* output envelopes. *For short, when no confusion may arise, we refer to $\mathcal{D}$ 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 $y_j + \mathtt{ub}$ is the identifier of an envelope in $B^K(e)$.
- *Privacy.* By running a computer $\mathcal{D}$ for f, the only *additional* information about f's inputs or outputs gained by the players consists of $R^K_{\mathcal{D}}$, the portion of the public history generated by $\mathcal{D}$'s execution. Privacy guarantees that this additional information is constant, thus the players neither learn anything about each other inputs or outputs nor receive any residual information. At the same time, in any execution the internal information of the device is the random string that can be generated with the same odds by a straight-line no-input simulator. Thus, the device also does not learn anything about the players' inputs or outputs.
- *Clean Operation.* Clean Operation guarantees that $\mathcal{D}$
  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 $\mathcal{D}$ are temporary, and will not exist in the final ballot set).
- *Simplicity.* Note that, since the public history generated by computer $\mathcal{D}$ is fixed, $\mathcal{D}$'s functions do not depend on public history R. Also, as we shall see, the private actions of the devices we construct depend only on at most 5 last records of H_0. Thus, one can interpret $\mathcal{D}$ as a simple automaton, that keeps in its internal memory last 5 private records, and reads the fixed string U record-by-record to find actions it has to perform.
- *Straight-line Simulators.* Our simulators are straight-line in the strictest possible sense. In essence, SIM is run independently many times, and each time outputs a random permutation in SYM_5 and its inverse. (The simulator is in fact called only after the device "privately opens a sequence of 5 envelopes" whose content is guaranteed —by construction— to be a random permutation of the integers 1 through 5.)

6 Three Elementary Ballot-Box Computers

In this section we first provide verifiably secure 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 $p \in \mathrm{SYM}_5$ to p^{-1}.
2. *Permutation Product,* mapping a pair of permutations $(p, q) \in (\mathrm{SYM}_5)^2$ to pq —i.e., the permutation of SYM_5 so defined: $pq(i) = p(q(i))$.
3. *Permutation Clone,* mapping a permutation $p \in \mathrm{SYM}_5$ to the pair of permutations (p, p).

Encodings. Note that the notion of a verifiably secure computer $\mathcal{D}$ 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 $\mathcal{D}$'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 s in SYM_5 with the 5-long string $s_1s_2s_3s_4s_5$, such that $s_j = s(j)$. 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 $cont_B(\sigma) \in SYM_5$.*

If σ is an envelope encoding of a permutation in SYM_5, we refer to this permutation by $\hat{\sigma}$. We consistently use lower-case Greek letters to denote envelope encodings.

Device Conventions. To simplify our description of a device $\mathcal{D}$ we adopt the following conventions.

- Rather than describing $\mathcal{D}$ as a sequence of K functions that, on input a public history R and a private history H_0, output a ballot-box action feasible for any global memory with public history R, we present $\mathcal{D}$ as a list of K actions $a^1, \ldots, a^K$ (to be performed no matter what the public history 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 $\texttt{ub} + 1$.
- 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 SYM_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.

6.1 A Verifiably Secure Computer for Permutation Inverse

Device $\mathcal{INV}_\sigma$

Input address: σ —an envelope encoding of a permutation in SYM_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 $(\sigma_\ell, \alpha_\ell)$.
(3) Ballotbox $A_1, \ldots, A_5$ to obtain $A'_1, \ldots, A'_5$.
(4) For $\ell = 1$ to 5: open super-envelope A'_ℓ to expose envelope pair (ν_ℓ, μ_ℓ).
(5) For $\ell = 1$ to 5: privately read and reseal ν_ℓ, and denote its content by $\hat{\nu}_\ell$. Set $\hat{\nu} = \hat{\nu}_1 \circ \cdots \circ \hat{\nu}_5$.
(6) For $\ell = 1$ to 5: make a new super-envelope B_ℓ containing the pair of envelopes (ν_ℓ, μ_ℓ).
(7) Secretly permute $B_1, \ldots, B_5$ according to $\hat{\nu}^{-1}$ to obtain $B'_1, \ldots, B'_5$.
(8) For $\ell = 1$ to 5: open super-envelope B'_ℓ to expose envelope pair (β_ℓ, ρ_ℓ). Set $\rho = \rho_1, \ldots, \rho_5$.
(9) For $\ell = 1$ to 5: open envelope β'_ℓ and denote its content by $\hat{\beta}_\ell$.
Output address: $37, 39, 41, 43, 45$.

Lemma 1. *For any 5-long address σ, $\mathcal{INV}_\sigma$ is a verifiably secure computer for permutation inverse, with input address σ and output address* $37, 39, 41, 43, 45$.

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

Correctness. Step 1 generates 5 new identifiers (increasing $\mathtt{ub}^0$ by 5). Step 2 binds together, in the same super-envelope A_ℓ, the ℓth envelope of σ and α. 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 $\hat{\sigma}$ and $\hat{\alpha}$, generating 5 new identifiers. Letting x be this secret permutation, Step 4 "puts on the table" the envelope encodings $\nu = \nu_1, \ldots, \nu_5$ and $\mu = \mu_1, \ldots, \mu_5$, where $\hat{\nu} = x\hat{\sigma}$ and $\hat{\mu} = x\mathcal{I} = x$, and generates 10 new identifiers. At the end of Step 4, both $\hat{\nu}$ and $\hat{\mu}$ are totally secret. In Step 5, however, the device learns $\hat{\nu}$ and reseals envelope encoding ν. Step 6 puts ν and μ back into super-envelopes $B_1, \ldots, B_5$, generating 5 new identifiers. In Step 7, the device secretly applies permutation $\hat{\nu}^{-1}$ to both $\hat{\nu}$ and $\hat{\mu}$, generating 5 new identifiers. The action of

Step 7 is feasible because $\hat{\sigma} \in \mathrm{SYM}_5$, thus $\hat{\nu} \in \mathrm{SYM}_5$. Step 8 "puts on the table" the envelope encodings β and ρ, where $\hat{\beta} = \hat{\nu}^{-1}\hat{\nu} = Id$ and $\hat{\rho} = \hat{\nu}^{-1}x$, and generates 10 new identifiers. Step 9 reveals contents of β, which are $\hat{\beta} = 12345$. Thus, $\rho = \mathtt{ub}^0 + 37, \mathtt{ub}^0 + 39 \ldots, \mathtt{ub}^0 + 45$; and $\hat{\rho} = \hat{\sigma}^{-1}x^{-1}x = \hat{\sigma}^{-1}$ as desired.

Privacy. It is clear that the public history generated by $\mathcal{D}$ is a fixed constant. And the very fact that the contents of β revealed in Step 9 are 1, 2, ..., 5 in the fixed order serves as a proof that the device had used the correct permutation to invert the contents of $\hat{\nu}$ and $\hat{\mu}$. Constructing the required simulator is also trivial, as the contents of $H_{0,D}$ are a random permutation $\hat{\nu}$ and its inverse. Thus, SIM consists of: (1) generating a random permutation $r = r_1 \ldots r_5 \in \mathrm{SYM}_5$; (2) for $\ell = 1$ to 5: writing a string r_ℓ; and (3) writing a string r^{-1}.

Clean Operation. Trivially follows by construction.

6.2 A Verifiably Secure Computer for Permutation Product

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

Input addresses: σ and τ —each an envelope encoding of a permutation in SYM_5.

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

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

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

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

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

(6) For $\ell = 1$ to 5: make a new super-envelope B_ℓ containing the pair of envelopes (ν_ℓ, μ_ℓ).

(7) Secretly permute $B_1, \ldots, B_5$ according to $\hat{\nu}^{-1}$ to obtain $B'_1, \ldots, B'_5$.

(8) For $\ell = 1$ to 5: open super-envelope B'_ℓ to expose envelope pair (β_ℓ, ρ_ℓ). Set $\rho = \rho_1, \ldots, \rho_5$.

(9) For $\ell = 1$ to 5: open envelope β'_ℓ and denote its content by $\hat{\beta}_\ell$.

Output address: $77, 79, 81, 83, 85$.

Lemma 2. *For any two, disjoint, 5-long addresses σ and τ, $\mathcal{MULT}_{\sigma,\tau}$ is a verifiably secure computer for permutation product, with input addresses σ and τ and output address* $77, 79, 81, 83, 85$.

Proof. To establish Correctness, note that envelopes α generated in Step 1 contain $\hat{\alpha} = \hat{\sigma}^{-1}$; contents of ν and μ in Step 4 are $\hat{\nu} = x\hat{\sigma}^{-1}$ and $\hat{\mu} = x\hat{\tau}$ for a random $x \in \mathrm{SYM}_5$; and contents of β and ρ in Step 8 are $\hat{\nu}^{-1}\hat{\nu} = \mathcal{I}$ and $\hat{\rho} = (x\hat{\sigma}^{-1})^{-1}x\hat{\tau} = \hat{\sigma}\hat{\tau}$. Privacy and Clean Operation trivially follow. By construction, the public history generated by $\mathcal{MULT}_{\sigma,\tau}$ is fixed, and the SIM has to generate a random permutation and its inverse twice (for $\mathcal{INV}$ in Step 1 and for Steps 5 and 7.)

6.3 A Verifiably Secure Computer for Permutation Clone

Device $\mathcal{CLONE}_\sigma$

Input address: σ —an envelope encoding of a permutation in SYM_5.

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

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

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

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

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

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

(7) For $\ell = 1$ to 5: make a new super-envelope B_ℓ containing the pair of envelopes $(\nu_\ell, \mu_\ell, \eta_\ell)$.

(8) Secretly permute $B_1, \ldots, B_5$ according to $\hat{\nu}^{-1}$ to obtain $B'_1, \ldots, B'_5$.

(9) For $\ell = 1$ to 5: open super-envelope B'_ℓ to expose envelope triple $(\delta_\ell, \psi_\ell, \rho_\ell)$.

(10) For $\ell = 1$ to 5: open envelope δ_ℓ.[13]

Output addresses: $92, 95, 98, 101, 104$ and $93, 96, 99, 102, 105$.

Lemma 3. *For any 5-long address σ, $\mathcal{CLONE}_\sigma$ is a verifiably secure computer for permutation clone, with input address σ and output addresses* $92, 95, 98, 101, 104$ *and* $93, 96, 99, 102, 105$.

7 General Verifiably Secure Computers

Recall that any finite function $f : \{0,1\}^a \to \{0,1\}^b$ can be easily (and quite efficiently) represented as a combinatorial circuit, and thus 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 $f : \{0,1\}^a \to \{0,1\}^b$ by utilizing these 4 basic computers. To this end, we must first decide how to encode binary strings and binary functions.

[13] Note that Steps 2–10, in essence, correspond to a protocol for permutation inverse that on input α produces two identical envelope encodings, each encoding $\hat{\alpha}^{-1}$.

Definition 8. *We define the* SYM_5 *encoding of a* k*-bit binary string* $x = b_1, \ldots, b_k$*, denoted by* $\bar{x}$*, to be* $\bar{b}_1 \cdots \bar{b}_k$*, where*

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

The SYM_5 *encoding is immediately extended to binary functions as follows:* $\bar{f}(\bar{x}) = \overline{f(x)}$ *for all* x*.*

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* $\overline{DUPLICATE}$ *with input address* σ*.*

Proof. Because $\mathcal{CLONE}$ duplicates any permutation in SYM_5, in particular it duplicates 12345 and 12453.

We thus proceed to build the other 3 basic computers

7.1 A Verifiably Secure Computer for $\overline{COIN}$

Device $\mathcal{COIN}$

(1) Create an envelope encoding α of $\mathcal{I}$ and an envelope encoding β of a.

(2) Make new super-envelopes A and B containing envelopes $\alpha_1, \ldots, \alpha_5$ and $\beta_1, \ldots, \beta_5$, respectively.

(3) Ballotbox A and B to obtain super-envelopes C and D.

(4) Open C to expose an envelope encoding γ. Destroy D.

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

Lemma 5. $\mathcal{COIN}$ *is a verifiably secure computer for* $\overline{COIN}$*, with no input address and output address* $15, \ldots, 19$*.*

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

Clean Operation is trivial, and Privacy straightforwardly follows by noting that the public history is fixed and there is no private history of the device generated by $\mathcal{COIN}$.

7.2 Verifiably Secure 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 [3] that the Boolean functions NOT and AND can be realized as sequences of group operations in SYM_5.[14] 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 $x \in \mathrm{SYM}_5$ as follows:

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

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

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

$$\overline{\neg b_1} = (x_1 a^{-1})^*, \quad \text{and} \quad \overline{b_1 \wedge b_2} = (x_1 \tilde{x}_2 x_1^{-1} \tilde{x}_2^{-1})'.$$

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 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.

[14] Note that neither $DUPLICATE$ nor $COIN$ can be realized in SYM_5, and thus one cannot compute arbitrary functions in SYM_5. Indeed, the result of [3] 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 SYM_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 SYM_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 SYM_5 into separate envelopes is crucial to our ability of performing general computation, and in a private way too.

7.3 Verifiably Secure Computers for Arbitrary Finite Functions

Theorem 1. *Every finite function has a verifiably secure computer.*

Proof. Let $f : \{0,1\}^a \to \{0,1\}^b$ be a finite function. Then (by properly ordering the "gates" of a combinatorial circuit for f) there exists a fixed sequence $C_f = F_1, F_2, \ldots, F_K$ such that:

- 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 $\mathcal{D}_i$ as follows:

$\mathcal{D}_i = \mathcal{COIN}$ if $F_i = COIN$;

$\mathcal{D}_i = \mathcal{CLONE}$ if $F_i = DUPLICATE$;

$\mathcal{D}_i = \mathcal{NOT}$ if $F_i = NOT$;

$\mathcal{D}_i = \mathcal{AND}$ if $F_i = AND$.

Let $\mathcal{D}$ be the concatenation of $\mathcal{D}_1, \ldots, \mathcal{D}_K$, with appropriately chosen input addresses, so that the lth input address of $\mathcal{D}_i$ matches the mth output address of $\mathcal{D}_j$ whenever the lth input bit of F_i is the mth output bit of F_j. Then, $\mathcal{D}$ is a ballot-box computer for f. In fact, $\mathcal{D}$'s correctness follows from the correctness of each computer $\mathcal{D}_i$. Furthermore, $\mathcal{D}$'s privacy follows from the fact that each $\mathcal{D}_i$ has a simulator SIM_i, and thus a simulator SIM for $\mathcal{D}$ can be obtained by executing (in order) $SIM_1, SIM_2, \ldots, SIM_K$. Finally, the clean operation of $\mathcal{D}$ follows from the clean operation of each $\mathcal{D}_i$.

Remarks.

- Note that our ballot-box computer $\mathcal{D}_f$ is "as efficient as" f itself. Indeed, the description of $\mathcal{D}_f$ is linear in the description of C_f. This is so because, letting $C_f = F_1, F_2, \cdots$, $\mathcal{D}_f$ 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 $\mathcal{D}_f$ is also linear in the time needed to run C_f.
- If $\mathcal{D}$ is executed on an initial global memory whose ballot set coincides with just the input envelopes, then the ballot set of $\mathcal{D}$'s final global memory of coincides with just the output envelopes.

8 The Input Stage (and the Output Stage)

Having described the concrete computation stage of ILM2 security, we just need to describe its input and output stages. The output stage is trivial implemented. In essence, the device publicly opens the sequence of envelopes containing the public output y, and hands over to each player i the sequence of envelopes containing y_i. For the latter, one must formally enrich the ballot-box model with the basic public action "Hand over envelope j to player i." We omit to do so formally in this extended abstract.[15]

In this section we thus limit ourselves to defining and implementing the input stage.

8.1 Verifiably Secure Input Committers.

Intuitively, a verifiably secure input committer $\mathcal{IC}$ is a protocol that enables each player i to give to the verifiably secure device $\mathcal{D}$ a separate sequence of envelopes S_i whose contents encode i's chosen input m_i so that: (1) in any execution of $\mathcal{IC}$ the contents of the envelopes in S_i properly encode m_i; and (2) only player i knows m_i, while the other players and the device learn only a fixed public history.

Since in the input stage the players must interact with the device, we start with formalizing the notion of a protocol (restricted to our needs) and then the notion of a committer. Afterwards, we proceed to construct a committer.

Definition 9. *A (tight) protocol $\mathcal{P}$ is a triple (K, PS, AF), where*

- *K, the* length *of the protocol, is a positive integer;*
- *PS, the* player sequence*, is a sequence of K integers each from 0 to n; and*
- *AF, the* action function*, is a mapping from $K \times \mathcal{R}$ —where $\mathcal{R}$ is the set of all feasible public histories— to sets of ballot-box actions such that, for all $k \in [1, K]$ and $R \in \mathcal{R}$, $AF(k, R)$ specifies for player $i = PS^k$ either a single action or a pair of actions as follows:*
  - ** a single action a^k if $i = 0$;*
  - ** a pair of* SecretPermute *actions $\{a_0^k, a_1^k\}$, where a_0^k and a_1^k permute the same ballots $j_0, j_1 \in I_B$ with, respectively, permutations* 12 *and* 21.[16]

[15] If we want to capture additional security desiderata, such as deniability, then we should ensure that the players privately read and destroy the envelopes they finally receive. To this effect, it may be advantageous to enrich the ballot-box model with the operation "flash the content to envelope j to just player i." (In essence this is the operation corresponding to raising a card facing down just in the direction of player i.) The device will then destroy the envelope flushed to i. Again, the destroy action is not essential, and can be simulated by a verifiable device by means of a fixed sequence of the other ballot-box actions. Additional desiderata can also demand the "simultaneous execution" of some of the actions.

[16] Note that the set of the current ballot identifiers I_B can be fully obtained from R. For more general protocols, the actions of player i are allowed to depend on H_i too.

If $a = a_b^k$, we refer to b as a's *hidden bit* and refer to a as the *0-action* (of the pair) if $b = 0$, and as the *1-action* otherwise.

For a given $\mathcal{P}$, let ℓ_i be the number of times player i is asked to act by $\mathcal{P}$, that is $\ell_i = \#\{k \in [1, K] \,:\, PS^k = i\}$. Let $z_{\mathcal{P}}$ be a profile of strings, such that z_i is an ℓ_i-bit string.

An *execution of* $\mathcal{P}$ with *associated profile* $z_{\mathcal{P}}$ on an initial feasible global memory (B^0, R^0, H^0) is a sequence of global memories $(B^0, R^0, H^0), \ldots, (B^K, R^K, H^K)$ such that $(B^k, R^k, H^k) = a^k(B^{k-1}, R^{k-1}, H^{k-1})$ for all $k \in [1, K]$, where:

- $a^k = AF(k, PS, gm^{k-1})$, if $PS^k = 0$; and
- $a^k = a_b^k \in AF(k, PS, gm^{k-1})$, where b is the mth bit of z_i, where $m = \#\{j \in [1, k] \,:\, PS^k = i\}$, if $PS^k = i$.

Since the execution of $\mathcal{P}$ in all respects is similar to the execution of a device, we will retain all the notation defined for executions of devices.

Definition 10. *Let $\mathcal{IC}$ be a ballot-box protocol, $(\bar{0}, \bar{1})$ a bit-by-bit encoding, and x an address profile, $x = x_1, \ldots, x_n$. We say that $\mathcal{IC}$ is a L-bit* verifiably secure input committer *for $(\bar{0}, \bar{1})$ with output address profile x if there exists a unique sequence U such that, for every execution e of $\mathcal{IC}$ with associated profile z, whose initial global memory is empty, the following three properties hold:*

1. Correctness: $\forall i \in [1, n], \quad H_i^K(e) = z_i \in \{0,1\}^L$ *and* $cont_{B^K(e)}(x_i) = \overline{z_i}$.

2. Privacy: $R^K(e) = U$ *and* $H_0^K(e) = \emptyset$.

3. Clean Termination: $I_{B^K(e)} = I_x$.

Remarks.

- *Correctness.* Correctness requires that, once $\mathcal{IC}$ is executed, the private history string of each player is of length L, and that the ballots corresponding to the output address profile $x_1, \ldots, x_n$ contain the bit-by-bit encoding of the players' intended inputs. This requirement has both syntactic and semantic implications. Syntactically, Correctness implies that each x_i is of length $5L$ and each element of x_i is the identifier of an envelope in $B^K(e)$. Further, it requires that the number of times each player acts in $\mathcal{IC}$ is equal to $5L$ and that the length of his private history is also $5L$. Semantically, Correctness implies that the envelopes of address x_i encode a message z_i freely chosen by player i alone. (I.e., the other players have no control over the value of z_i.) This is so because the envelopes in x_i are guaranteed to contain the bit-by-bit encoding of the final private record of player i. Recall that i's private history, H_i, grows, by a bit at a time, in only one case: when i himself secretely chooses one of two complementary actions (in our physical interpretation, when i temporarily holds two envelopes behind his back, and then returns them in the order he chooses). This is an "atomic" (in the sense of "indivisible") choice of i, and therefore the other players have no control over it.
- *Privacy.* Privacy requires that, at each round k of $\mathcal{IC}$, the public information available to the acting player always consists of the fixed sequence

U^{k-1}, no matter what are the intended inputs. Thus, while Correctness implies that any control over the choice of a player's message solely rests with that player, Privacy implies that all messages are independently chosen, as demanded in the ideal normal-form mechanism, and totally secret at the end of the committer's execution.

Privacy thus rules out any possibility of signaling among players during the execution of a committer.

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

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

8.2 Our Verifiably Secure Committer

Protocol $Commit_L$

For player $i = 1$ to n DO: For $t = 1$ to L and bit b_{it} Do:

(1) Create an envelope encoding $\alpha^{(i,t)}$ of permutation $\mathcal{I} = 12345$.

(2) Create an envelope encoding $\beta^{(i,t)}$ of permutation $a = 12453$.

(3) Make a new super-envelope $A^{(i,t)}$ containing envelopes $\alpha_1^{(i,t)}, \ldots, \alpha_5^{(i,t)}$.

(4) Make a new super envelope $B^{(i,t)}$ containing envelopes $\beta_1^{(i,t)}, \ldots, \beta_5^{(i,t)}$.

(5) Player i secretly permutes $A^{(i,t)}$ and $B^{(i,t)}$ according to 12, if $b_{it} = 0$, and to 21, otherwise, to obtain the super-envelopes $C^{(i,t)}$ and $D^{(i,t)}$.

(6) Open $C^{(i,t)}$ to expose envelopes $\gamma_1^{(i,t)}, \ldots, \gamma_5^{(i,t)}$. Set $\gamma^{(i,t)} = \gamma_1^{(i,t)}, \ldots, \gamma_5^{(i,t)}$.

(7) Destroy $D^{(i,t)}$.

Output Addresses: For each $i \in N$, the sequence $x_i = \gamma^{(i,1)}, \ldots, \gamma^{(i,L)}$.

Lemma 7. *Protocol $Commit_L$ is an L-bit verifiably secure committer for the SYM_5 encoding.*

PROOF. At the end of Step 3, $A^{(i,t)}$ contains a sequence of 5 envelopes encoding the identity permutation $\mathcal{I}$, and, at the end of Step 4, $B^{(i,t)}$ contains a sequence of 5 envelopes encoding permutation a. Thus, recalling that in the SYM_5 encoding $\mathcal{I} = \bar{0}$ and $a = \bar{1}$, at the end of Step 5, $C^{(i,t)}$ contains an envelope encoding of $\bar{b}$ if player i "chooses the bit b". Thus, at the end of Step 6, the content of address x_i is $\overline{z_i}$, where H_i is a binary string of length L, as demanded by Definition 10. All other properties are trivially established.

References

1. Robert Aumann. Subjectivity and correlation in randomized strategies. *J. Math. Econ.*, 1:67–96, 1974.
2. Imre Bárány. Fair distribution protocols or how the players replace fortune. *Mathematics of Operation Research*, 17:327–341, May 1992.
3. David Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. In *Proceedings of STOC*, 1986.
4. Michael Ben-Or, Shafi Goldwasser and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of STOC*, 1988.
5. Elchanan Ben-Porath. Correlation without mediation: Expanding the set of equilibria outcomes by "cheap" pre-play procedures. *Journal of Economic Theory*, 80:108–122, 1998.
6. Ran Canetti. Universally composable security. A new paradaigm for cryptographic protocols. *Proceedings of FOCS*, 2001.
7. Richard Cleve. Limits on the Security of Coin Flips When Half the Processors are Faulty. *Proceedings of STOC*, 1986.
8. Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *Proceedings of CRYPTO*, volume 1880 of *LNCS*. Springer-Verlag, 2000.
9. Yevgeniy Dodis and Silvio Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *Proceedings of CRYPTO*, volume 1880 of *LNCS*. Springer-Verlag, 2000.
10. Dino Gerardi. Unmediated communication in games with complete and incomplete information. *Journal of Economic Theory*, 114:104–131, 2004.
11. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of STOC*, 1987.
12. Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation. In *Proceedings of STOC*, 2004.
13. Sergei Izmalkov, Matt Lepinski, and Silvio Micali. Rational secure function evaluation and ideal mechanism design. In *Proceedings of FOCS*, 2005.
14. Eyal Kushilevitz, Yehunda Lindell, and Tal Rabin. Information-Theoretically Secure Protocols and Security under Composition. In *Proceedings of STOC*, 2006.
15. Matt Lepinksi, Silvio Micali, Chris Peikert, and Abhi Shelat. Completely fair SFE and coalition-safe cheap talk. In *Proceedings of PODC*, 2004.
16. Amparo Urbano and Jose E. Vila. Computational complexity and communication: Coordination in two-player games. *Econometrica*, 70(5):1893–1927, 2002.
17. Andrew Yao. Never published. The result is presented in Lindell and Pinkas (2004): "A proof of Yao's protocol for secure two-party computation," available at: *http://www.cs.biu.ac.il/ lindell/abstracts/yao_abs.html*.