The Security Power of the Ballot Box

Matt Lepinski and Silvio Micali

February 2, 2005

Abstract

We show that any function f can be securely evaluated by a protocol with ballots and a ballot box. That is, n mutually suspicious players, each player i possessing a secret input x_i, can use ballots and ballot boxes to jointly evaluate $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, so that (no matter how many players may collude and deviate from their prescribed instructions, and no matter how long they compute!) each player i learns exactly y_i with the same privacy and correctness as if all players privately handed their secret inputs to a trusted party, who privately evaluates f and privately returns y_i to each player i.

Our protocol is (1) efficient, (2) enjoys perfect privacy, (3) guarantees perfect correctness, (4) is universally composable in the sense of [DoMi00] and [Can01], and (5) is always collusion-free in the sense of [LMS05], even for games with secret actions.

1 Main Theorem

For any function f, mapping n private inputs to a single public output, there exists an n party protocol for a ballot-box network with public opening that securely evaluates f.

We now procedure to define (1) a ballot-box network, (2) a ballot-box network with public opening and (3) the notion of securely evaluating a function. In a subsequent version of this work, we will provide a proof of our main theorem, generalize our results to functions with n inputs and n outputs and explore applications to economic mechanisms.

2 Communication Models

2.1 Ballot-Box Networks

Intuition

We envisage a group of players, seating far apart around a large table, communicating in two ways. The first way is via *broadcasting:* a player stands up and loudly utters a given message. The second way is via identical, opaque *envelopes* (and *super-envelopes*) and a *ballot box*.

Informally, a player can choose a message, write it on a piece of paper and seal it into a new, empty envelope. So long as it is not opened, the envelope totally hides and guarantees the integrity of its content. Only the owner of a sealed envelope can open it, either *privately* (in which case — though all other players are aware that he is opening it—- he will be the only one to read its content) or *publicly* (in which case all players will learn its content). A player *owns* a sealed envelope if it is physically close to him. By definition, the player originally sealing a new envelope owns it. After that, ownership of an envelope can be transferred to another player by tossing it to him.

A player can also publicly put up to 4 of his envelopes into a new super-envelope E, in which case none of the resulting "sub-envelopes" can be opened before E. All super-envelopes are again opaque and identical among each other, but have a slightly larger size than (ordinary) envelopes. A super-envelope E thus tightly packs its sub-envelopes, so that their relative order (counting —say— from E's front) does not change when E is moved about. The rules of ownership for super-envelopes are the same as for envelopes. There is, however, only one possible way for a player i to open a super-envelope E of his: namely, all players become aware that i has opened E, and E's sub-envelopes become "exposed" again, and can thus be manipulated (e.g., opened or transferred) individually. Such sub-envelopes always guarantee the integrity and privacy of their contents.

Envelopes and super-envelopes always stay above the table and their transfers are always tracked by the players. The players can thus "mentally assign" to each envelope or super-envelope an identifier, j, insensitive to any possible change of ownership. The only exception is when a player i publicly puts some of his envelopes or super-envelopes into a ballot box: when they are taken out, their contents will remain unchanged and private, but their identities are randomly permuted, in a way that is unpredictable to all players.

In essence, by putting on the table, opening, transferring and "ballot-boxing" envelopes and super-envelopes, we obtain a communication system having a global memory different portions of which are observable by different players.

Formalization

We denote by A^L the Cartesian product of a set A with itself L times; by Σ the set of all finite binary strings; by `#` a special symbol not in Σ; by S_L the group of all permutations of L elements; by $x := y$ the operation that assigns value y to variable x; by $rand(X)$ the function that selects a random element (uniformly and independently) from a finite set X.

> **Definition:** An *envelope* is a triple (i, j, c), where i is a player, j a positive integer, and c a finite binary string. A *super-envelope* is a triple (i, j, c), where again $i \in N$ and $j \in \mathbb{N}$, but $c \in \Sigma^L$, $L > 1$, is a tuple of strings. A *ballot* is either an envelope or a super-envelope. If (i, j, c) is ballot, we refer to i as the ballot's *owner*, to j as the ballot's *identifier*, and to c as the ballot's *content*. A set of ballots is *well-defined* if, for each $j \in \mathbb{N}$, there exists at most one ballot with identifier j. By the expression "ballot j" we mean the unique ballot, if any, having identifier j. To emphasize that ballot j actually is an envelope (super-envelope) we may use the expression "envelope j" ("super-envelope j").

> **Definition:** A *ballot-box network* consists of a set of global variables, the *network memory,* modifiable solely through a specific set of operations, the *network operations.*
>
> The variables of the network memory are: a well-defined set of ballots B; a public history H, readable by all players; and, for each player i, a *private history of* i, h_i, readable only by i. (Variable H is a publicly observable transcript of all network operations so far; it consists of a list of items separated by `#`. Variable h_i is i's private transcript of all network operations so far.)
>
> The network operations (defined via the auxiliary variable max, the highest identifier of a ballot in B) are:
>
> – *player i broadcasts m:* if $m \in \Sigma$, then $H := H\#\ \texttt{broadcast}, i, m$

- *i makes a new envelope with public content c:* if $c \in \Sigma$, then $max := max + 1$; $B := B + \{(i, max, c)\}$; $h_i := h_i\texttt{\#}\ (i, max, c)$; and $H := H\texttt{\# NewPublicEnvelope}, (i, max, c)$.
- *i makes a new envelope with private content c:* if $c \in \Sigma$, then $max := max + 1$; $B := B + \{(i, max, c)\}$; and $H := H\texttt{\# NewPrivateEnvelope}, i, max$.
- *i publicly opens envelope j:* if $(i, j, c) \in B$ and $c \in \Sigma$, then $B := B - \{(i, j, c)\}$ and $H := H\texttt{\# open}, (i, j, c)$.
- *i privately opens envelope j:* if $(i, j, c) \in B$ and $c \in \Sigma$, then $B := B - \{(i, j, c)\}$; $h_i := h_i\texttt{\#}\ (i, j, c)$; and $H := H\texttt{\# Open}, i, j$.
- *i makes a new super-envelope from envelopes $j_1, \ldots, j_L$:* if $(i, j_1, c_1), \ldots, (i, j_L, c_L) \in B$, then $B := B - \{(i, j_1, c_1), \ldots, (i, j_L, c_L)\}$; $max := max + 1$; $B := B + \{(i, max, (c_1, \ldots, c_L)\}$; $H := H\texttt{\# NewSuper}, i, j_1, \ldots, j_L$.
- *i opens super-envelope j:* if $(i, j, (c_1, \ldots, c_L)) \in B$, then $B := B - \{(i, j, (c_1, \ldots, c_L))\}$; $B := B + \{(i, max + 1, c_1), \ldots, (i, max + L, c_L)\}$; $max := max + L$; and $H := H\texttt{\# OpenSuper}, i, j$.
- *i transfers ballot j to k:* if $(i, j, c) \in B$, then $max := max + 1$; $B := B - \{(i, j, c)\} + \{((k, max, c)\}$; $H := H\texttt{\# Transfer}, i, j, k$.
- *i ballot-boxes $j_1, \ldots, j_L$:* if $(i, j_1, c_1), \ldots, (i, j_L, c_L) \in B$, then $B := B - \{(i, j_1, c_1), \ldots, (i, j_L, c_L)\}$; $\pi := rand(S_L)$; $B := B + \{(i, max{+}1, c_{\pi(1)}), \ldots, (i, max{+}L, c_{\pi(L)})\}$; $max := max{+}L$; $H := H\texttt{\# BallotBox}, i, j_1, \ldots, j_L$.
- *i reads and reseals envelope j:* if $(i, j, c) \in B$ and $c \in \Sigma$ then $h_i := h_i\texttt{\# (i,j,c)}$; $H := H\texttt{\# Reseal}, i, j$.
- *i publicly permutes identifiers $j_1, \ldots, j_L$ by π:* if $(i, j_1, c_1), \ldots, (i, j_L, c_L) \in B$, then $B := B - \{(i, j_1, c_1), \ldots, (i, j_L, c_L)\}$; $B := B + \{(i, max + 1, c_{\pi(1)}), \ldots, (i, max + L, c_{\pi(L)})\}$; $max := max + L$; $H := H\texttt{\# PublicPermute}, i, j_1, \ldots, j_L, \pi$.
- *i aborts:* $H := H\texttt{\# Abort}, i$.

Network operations are executed one at a time, and without any interruption. Immediately after an execution, all players actually read the modified variables available to them.

By the expression *i makes a new envelope j with private (or public) content c* we mean that i makes a new envelope and that the identifier assigned to it actually is j. Similarly, by i *makes a new super-envelope* j *from envelopes* $j_1, \ldots, j_L$ *with identifier* j we mean that j actually is the identifier assigned to the new super-envelope made by i. Finally, we will use the expressions i *opens super-envelope* j *to expose envelopes* $k_1, \ldots, k_L$ and i *ballot-boxes* $j_1, \ldots, j_L$ *to get ballots* $k_1, \ldots, k_L$ to mean that identifiers $k_1, \ldots, k_L$ are the identifiers assigned to the ballots created by the opening or ballot-boxing operation.

Remarks

Notice that B remains well-defined after each network operation. Also notice that, at any point, the highest identifier of a ballot in B is bounded by the total number of ballots created up to that point.[1]

Our formalization of a ballot-box networks captures the salient features of "ownership" and "opening" of our intuitive model. No player can open a ballot he does not own. When a super-envelope is created from a set of envelopes, the latter are removed from the ballot set, and thus cannot be opened by anyone. Finally, when a super-envelope $(i, j, (c_1, \ldots, c_L))$ is opened by i, new envelopes with owner i and respective contents $c_1, \ldots, c_L$ are created. This is consistent with our intuitive description, where ownership corresponds to physical proximity: if super-envelope j is close to player i and, due to this proximity, it is indeed opened by him, then the envelopes exposed by opening super-envelope j will also be close to i.

To simplify the description of our protocols, we provide a redundant set of network operations. For example, to broadcast m a player can create a new envelope with contents m and then publicly open it. As for another example, the read-and-reseal operation could be implemented —although in a convoluted way— using only other network operations.

Our formalization of super-envelopes, and ballot boxes is slightly more general than needed in this paper: our protocols do not require super-envelopes to contain at more than 4 sub-envelopes or ballot-boxes to randomize more than 5 ballots, nor the identification of the player causing an abort.

The abort operation merely alerts all players by placing a special symbol in the public history, but does not effect the future network operations of

[1]Therefore our way to ensure that no two ballots have the same identifier has no negative effect on the existence of "polynomial-time" protocols. (If, instead, each new ballot identifier were chosen to be twice as long as the previous one, a protocol using k ballots would require exponentially many (in k) elementary operations —which certainly include the reading or writing of each identifier bit.)

"bad players." This is again consistent with our intuitive model: for instance, a player leaving the table cannot stop other players from opening their own ballots or making new ones. (In our protocols, when the symbol `Abort` appears in the public history, a "good" player takes no further action. But "bad" players may continue to perform network operations on the current network memory. Thus our protocols are designed so that, if an abort occurs, bad players cannot gain undue information by opening their own ballots.)

2.2 Ballot-Box Networks with Simultaneous Public Opening

We must augment our standard ballot-box network with on additional operation in order to securely evaluate a function with a single public output in such a way that either all players learn the output or else no one learns anything. The operation we add is that, if all players agree, then a specified set of envelopes will be publicly opened. We now describe this operation formally:

> **Definition:** A *ballot-box networks with simultaneous public opening* is a ballot-box network augmented with the following operation.
>
> – *All open envelopes* $j_1, \ldots, j_L$*:* If $(i_1, j_1, c_1), \ldots, (i_L, j_L, c_L)$ are envelopes in $\in B$, and H ends in the string `# broadcast`, $1, (j_1, \ldots, j_L)$ `# ... # broadcast`, $n, (j_1, \ldots, j_L)$, then $H := H\texttt{\#}(c_1, \ldots, c_L)$.

We think of this operation as being performed with the aid of a weakly trusted party who collects the envelopes to be opened and then publicity reveals their contents. Notice that this party is not trusted with any secrets, he is only trusted to open a set of envelopes and real their contents.

3 The Notion of Securely Computing Common-Output Functions

In this section we wish to define what it means for n players to compute securely a function f from n private inputs to a single, public output (in any of our envisaged communication networks).

3.1 Intuition

Informally, a protocol specifies the strategies, $P_1, \ldots, P_n$, for good players to use for communicating in the underlying network. If each player i started with private input x_i and all players are good —-i.e., follow their own strategies— $(P_1(x_1), \ldots, P_n(x_n))$ specifies an execution of a protocol.

Difficulty of Secure Evaluation without Bad Players. Notice, however, that designing a protocol P for securely evaluating a function $y = f(x_1, \ldots, x_n)$ is hard, even if all players stick to communicating according to their prescribed strategies. In fact "forget what you have seen after termination except the result" is not a reasonable instruction —certainly it does not an operation of our communication models! Consequently, players who send messages as told, can easily evaluate f correctly, but can they as easily guarantee the desired privacy? For instance, assume that each x_i represents the price that player i is willing to pay for a given item, and the player wish to implement a second-price auction —i.e., that f is the function returning the second highest price and the identity of the highest bidder. Then it is trivial to find strategies $P_1, \ldots, P_n$, so that f is correctly evaluated, but much harder to ensure that the players, seeing their own transcripts of the communication cannot deduce more that the auction desired outcome!

Secure Evaluation with One Bad Player. This task become more daunting if some players will deviate from their prescribed strategies. Traditionally, Game Theory focuses on *single-player deviations.* For instance, a single player, i, may repudiate P_i and use instead a deviating strategy P_i'. In this case, our task becomes that of designing an n-tuple of strategies, P, such that, for any possible player i and strategy P_i', (P_i', P_{-i}) continues to be a correct and private evaluation of f.

The Problem of Bad Inputs. Assume now that there is only one bad player, but what is his input? For a good player, i, his input is x_i. A bad player j, however, might ignore his input in which case we have no hope of computing a function of this input. On the other hand, we cannot say that a bad player j has no input or has an input which is pinned down after the protocol is complete. If a bad player is not required to commit to some input early in the protocol then he might be able to produce unacceptable outcomes. For example, in an auction, a bad player might get the auction to always output "player j wins and pays price $\max_{i \neq j}(x_i)$. This is, of course, a valid output of the function if j had bid $\max_{i \neq j}(x_i) + 1$ but surely it is unacceptable to always produce this outcome when player j should not know the value $\max_{i \neq j}(x_i)$.

3.2 Formalization

PROTOCOLS AND ADVERSARIES. In all our protocols players are deterministic (the only source of randomness comes from their use of the ballot box). Therefore we slightly simplify our definition of a protocol so as to envisage deterministic strategies for the good players.

We envisage that one player at a time is active in taking an action: for instance, the order of activity could be $1, \ldots, n, 1, \ldots$ If player 1 should act based on the action of player 2, however, this order of activity is inefficient, since all other players would need to take the "empty action" to enable the desired dependency. For efficiency sake, we thus let a protocol specify the exact sequence of players who will be active in its executions. If a protocol envisages k operations, it will specify the sequence of k players performing them.

An adversary is an algorithm that corrupts and controls some of the players. We explicitly specify the adversary be probabilistic, and thus, to guarantee a complete description of an execution, we let the adversary algorithm produce the coin tosses it uses in its computations.

> **Definition:** An n-player, k-operation *protocol* P consists of an n-tuple of *strategies,* and a k-long *sequence of players.* We let P_i denote ith strategy of P, and PS the sequence of players of P. On any input, P_i produces either (1) a network operation op_i for player i, or (2) the special symbol `halt` and an *output* $out_i \in \Sigma \cup \{\texttt{abort}\}$.
>
> An *adversary* A is a probabilistic strategy that, on any input X, produces (1) a value v and the sequence of coin tosses R used to compute v from X.

EXECUTING PROTOCOLS AND ADVERSARIES. When a protocol is executed with an adversary, all players begin with private input — the values on which they want to compute a function — and auxiliary inputs – which represents any additional information a player may have from previous protocols. "Good" players following the protocol strategy ignore their auxiliary input, but when the adversary corrupts a good player it learns the player's auxiliary input and may make use of that information. During an execution, an adversary view h_A is maintained which contains all information available to the adversary. In an execution, the adversary gets a chance to corrupt players, then the first player in the protocol's player sequence performs an operation, the adversary gets another chance to corrupt players and the protocol continues with the next player in the player sequence selecting an operation. This continues until the player sequence is reached and all players have halted.

Definition: An execution of P with inputs $(x_1, \ldots, x_n)$ and auxiliary inputs $(a_1, \ldots, a_n)$ and A with auxiliary input α and set of corrupted players $C \subset N$ is generated by processing —one at a time— every element in *PS*, starting with an empty network memory and an initial adversary history $h_A = (\alpha, C, x_C)$.

Processing an element of *PS* consisting of (an occurrence of) player i is done in two phases. In the first, A corrupts as many additional players as he desires. In the second, player i selects the network operation to be executed. That is, i is processed by executing the following three steps:

Step 1: The adversary history is updated so as to include the right portions of the network memory: $h_A := h_A \texttt{ \# } h_c, H$. Then A is run on the updated h_A to produce a value v with corresponding coins R, and the adversary history is further updated so as to include the coins just used: $h_A := h_A \texttt{\#} R$. As long as v is the identity of a player j, then i is added to the set of corrupted players, $C := C \cup \{j\}$, i's private and auxiliary inputs and private history are given to the adversary, $h_A := h_A \texttt{\#}\ (a_j, x_j, h_j)$, and A is run again on the updated h_A.

Step 2: If $i \in C$, then A is run on h_A to produce a value v with corresponding coins R, and the adversary history is updated so as to include these coins: $h_A := h_A \texttt{\#}\ R$. If v is a network operation for i then the operation is immediately performed so as to update the network memory.

Step 3: If $i \notin C$, unless P_i has previously output `halt`, P_i is run on input (x_i, h_i, H) and (1) if P_i produces a network operation op_i, then op_i is immediately executed so as to modify the current network memory, otherwise (2) if P_i produces `halt` then the current public history and the private history of i are appended to a_i: $a_i := a_i \texttt{\#} H, h_i$.

After all elements of *PS* have been processed, Step 1 is executed one more time (to give the adversary an opportunity to corrupt additional players and gain additional information).

EXECUTION NOTATION. In an execution of a protocol P, we say that *player i halts* when P_i outputs `halt`, and that the *protocol halts* when all players halt.

We say that an execution of P has *aborted (by operation m)* if a good player outputs `abort` (before the processing of the mth element of PS has ended).

If e is an execution of P and A, then

- $h_A(e)$ shall denote the *final history of A:* the value of h_A after the final execution of Step 1;
- $h_A^m(e)$ shall denote the value of h_A when operation $m+1$ is about to be executed;[2]
- $C(e)$ shall denote the *final corrupted set*: C's value after the final execution of Step 1.
- $C^m(e)$ shall denote the set of corrupted players when operation $m+1$ is about to be executed;
- $H(e)$ shall denote the *final public history in e;* and
- $h_i^m(e)$ shall denote the private history of player i in e after processing the mth element of PS;
- $h_i(e)$ shall denote the *final private history of player i* in e: i.e., $h_i(e) = h_i^k(e)$.

Effective Inputs and Outputs. The effective inputs of the players are the values upon which the function is actually computed and depends on the network operations executed before some round m. The effective input function determines a player's effective input based on his private history. We require that good players have effective inputs equal to their private input x_i but a corrupted player might select a different value as his effective input. An effective output is the public output actually computed by the protocol. The effective output function determines the effective output based on the final public history.

> **Definition:** Let P be a k-operation, n-party protocol, $IN = (IN_1, \ldots, IN_n)$ an n-tuple of functions. We say that IN is an *effective input function* for P if there exists a natural number $m < k$ such that, in any execution e of P with an adversary A
>
> - If e has aborted by the mth operation, then $IN_i(h_i(e)) = IN_i(h_i^m(e)) = \perp$ for all players i;
> - Else, $x_i = IN_i(h_i(e) = IN_i(h_i^m(e))$ for all good players i.

[2] I.e., the value of h_A at the end of Step 1 of the processing of element $m+1$ of PS

We say that OUT is an *effective output function* for P if, in any execution e of P with an adversary A that has aborted, $OUT(H(e)) = \perp$.

SECURE PROTOCOLS. A protocol securely evaluates a function if no matter what a malicious adversary does (1) he cannot prevent the function from being correctly computed unless he aborts the protocol; (2) prior to the final public opening, he has absolutely no information about the honest player inputs except what can be derived from the corrupted player inputs and auxiliary inputs; and (3) even after the the final public opening, the only information he has about honest player inputs is what can be derived from the corrupted player inputs and the public output.

Definition: We say that a protocol P in a ballot-box network with simultaneous public opening *securely evaluates* a function f if there exists an effective input function IN and an efficiently computable *output function* OUT such that for any adversary A corrupting at most $n-1$ players and for any distribution D over tuples $(x_1, \ldots, x_n, a_1, \ldots, a_n, \alpha)$, in a random execution e of P and A with inputs drawn from D the following three properties hold:

- If e has not aborted then $f(IN_1(h_1(e)), \ldots, IN_n(h_n(e))) = OUT(H(e))$;
- $I(x_{-C^{k-1}(e)}; h_A^{k-1}(e) \mid \langle\, x_{C^{k-1}(e)}, a_{C^{k-1}(e)}, \alpha \,\rangle\,) = 0$;
- $I(x_{-C(e)}; h_A(e) \mid \langle\, x_C, a_{C(e)}, \alpha, OUT(H(e)) \,\rangle\,) = 0$.

Remarks

Above we state our definition in terms of mutual information. However, we could equivalently state our definition in terms of conditional entropy as follows:

$$\begin{aligned} H(x_{-C^{k-1}(e)} \mid \langle\, x_{C^{k-1}(e)}, a_{C^{k-1}(e)}, \alpha \,\rangle\,) &= H(x_{-C^{k-1}(e)} \mid \langle\, h_A^{k-1}(e), x_{C^{k-1}(e)}, a_{C^{k-1}(e)}, \alpha \,\rangle\,) \\ H(x_{-C(e)} \mid \langle\, x_{C(e)}, a_{C(e)}, \alpha, OUT(H(e)) \,\rangle\,) &= H(x_{-C(e)} \mid \langle\, h_A(e), x_{C(e)}, a_{C(e)}, \alpha, OUT(H(e)) \,\rangle\,) \end{aligned}$$

References

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd FOCS*, 2001.

[DoMi00] Yevgeniy Dodis and Silvio Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *Proc. of Crypto 2000*, 2000.

[LMS05] Matt Lepinski, Silvio Micali and Abhi Shelat. Collusion-Free Protocols. In *Proc. 37th STOC*, 2005.