

# Output Privacy in Secure Multiparty Computation

Emmanuel Bresson<sup>1</sup>, Dario Catalano<sup>2</sup>,  
Nelly Fazio<sup>3</sup> Antonio Nicolosi<sup>3</sup> and Moti Yung<sup>4</sup>

<sup>1</sup> Cryptology Department, CELAR, France. [emmanuel.bresson@polytechnique.org](mailto:emmanuel.bresson@polytechnique.org)

<sup>2</sup> CNRS - École Normale Supérieure, Paris, France. [dario.catalano@ens.fr](mailto:dario.catalano@ens.fr)

<sup>3</sup> New York University, USA. {[fazio](mailto:fazio@cs.nyu.edu), [nicolosi](mailto:nicolosi@cs.nyu.edu)}@cs.nyu.edu

<sup>4</sup> Columbia University & RSA Labs, USA. [moti@cs.columbia.edu](mailto:moti@cs.columbia.edu)

**Abstract.** In this work, we define the notion of *private-output multiparty computation*: this newly revised notion, which encompasses (as a particular case) the classical definition, allows players to jointly compute the output of a function such that no information about the *outputs* leaks to an adversary controlling some of the players (apart from what follows from the description of the function itself). We formally verify that no function can be output-privately computed if the adversary gets full access to corrupted players’ memory. However, if the adversary controls only part of the *state* of corrupted players, any function can be output-privately computed, assuming that enhanced trapdoor permutations exist. We prove security is preserved under sequential composition.

## 1 Introduction

Secure multi-party computation (MPC) [16, 11] enables a set of mutually mistrusting parties to jointly compute a function, while keeping their inputs private. The power of the framework stems from the fact that under corruption of some parties (even by a malicious adversary) and according to various settings, it is possible to compile any poly-sized function into a protocol that maintains input privacy. It is thus a strong tool in solving very general cryptographic problems (*cf. e.g.*, [11, 2, 6, 7, 5, 9]).

In all the standard flavors, however, the privacy concern (*cf. e.g.* [7, 1]) is about the parties’ *inputs*, and no assurance is provided about the confidentiality of the *output(s)*. Disclosure of the output(s) may indeed have serious consequences on the security: *e.g.*, when the computed output is a secret key; or if there is a need to reveal the output only at a certain point, in “real time,” and not before.

One could think about computing the given function using standard MPC techniques, while keeping the output “distributed” among the parties. This would seemingly protect against outsider adversaries. Such *ad-hoc* solution, however, only dodges the problem: how can we prove that this is “secure” if current definitions *do not model* the desired security concerns? Clearly a more structured approach would be preferable. Though this is mostly a definitional effort, we believe that shedding light on it is important to improve our understanding of a central cryptographic tool such as MPC.

---

Research conducted while visiting École Normale Supérieure, Paris.

Research conducted in part while visiting École Normale Supérieure, Paris.

**Difficulties and main ideas.** Due to the fact that the output of one player may coincide with that of other players (and correctness should guarantee that this is the case), a single corruption might reveal to the adversary the output of *all* players. Intuitively, this makes the notion of output-privacy hard to capture with respect to traditional *active (misbehaving) adversaries* while security against “outsiders” should be achievable. In practice, however, security against outsiders is of low interest. Consequently, an important issue is to determine the highest attainable level of protection: the definition should be strict enough to encompass the impossibility issue, but also sufficiently general to leave room for “interesting” adversaries to consider.

We start from the following observation. Whenever the adversary corrupts a player, its input, its random coins, and the messages exchanged with the other parties, are already enough to derive the value of the global output. Hence, to there be any hope of protecting such value, the power of the adversary must be reduced by postulating that it cannot access (part of) these three quantities, *even for the corrupted players*.

Since communication is public, removing the *received* messages from the adversary’s view seems too strong. Similarly, the input is often decided by a higher-level protocol, and so is likely to be known to the adversary. Thus, the most natural way to restrict the adversary’s view is to limit her access to the controlled parties’ randomness and outputs. Establishing how and when to limit the adversary’s access to this randomness leads to new classes of adversaries of increasing capabilities (*i.e.*, the more randomness the adversary is allowed to access, the more powerful she is). This allows us to prove that, under the assumption that enhanced trapdoor permutations [10] exist, every function is output-privately computable even in the presence of an adversary with, basically, the sole restriction that she cannot see the global output computed by the players.

## 2 A Model for Private-Output Computation: The Two-Party Case

Our security definitions follow the real *vs.* ideal methodology, [3]. Being interested in guaranteeing the global output’s privacy, we define real/ideal worlds as close as possible to their counterparts in the standard setting, but ensure that an adversary cannot extract information about the function’s global output.

**Preliminaries.** In the case of global-output multi-party computation, functionalities have a *global* output, which should be obtained by all the parties, along with *n local* outputs, one for each participant. We name a *Global-Output Multi-party Functionality* any functionality of the form:  $f : \mathbb{N} \times (\{0, 1\}^*)^n \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times (\{0, 1\}^*)^n$ . Informally, a global-output multi-party functionality  $f$  is *t-non-trivial*, with respect to a given distribution  $\mathcal{D}$ , if the global output is not entirely determined by the specification of  $f$  and the view of up to  $t$  inputs sampled from  $\mathcal{D}$ . In this work we focus on the common output.

### 2.1 The Real Model: Overview

As said above, the usual adversarial behaviors (*eavesdropping, honest-but-curious, active*) form a too restrictive classification for defining a notion of output-privacy. When

there exists a common output to both parties, we must assume that each party holds a certain level of trust on the other party (*e.g.*, that they are both interested in obtaining the correct outcome). However, such trust does not refer to the other party’s computing environment. A closer look leads us to consider the following components: (1) the *application context* which provides the input and obtains the outputs, (2) the *control wrapper* which activates the party’s code with (possibly altered) input and gives back a (possibly altered) *local* output — the *global* output, however, is communicated directly to the application context, and is thus never known to the control wrapper, (3) each party’s code, which is split into two Turing machines: the *driver* (in charge of high-level operations) and the *secure device*, which has *limited* capabilities, but carries out only the most sensitive computations.

**Modeling adversaries.** The control wrapper can be thought as used by an application to invoke crypto library code (the “driver”). Under normal circumstances, the control wrapper is just a dummy interface. However, a malicious software (such virus) could successfully subvert the mechanism; it is thus reasonable to consider “partial” insider adversaries who are able to take over it. We refer to such adversaries as *input-/output-controlling*, *communication-halting*<sup>1</sup>. A stronger adversary may replace the code for the driver itself. We refer to it as *state-controlling*. Notice, however, it is still restricted to use the interface provided by the tamper-resistant device. Such restrictions are dropped for the case of *active* adversary, which obtains full control over the corrupted party. To summarize, we have the following adversaries (in order of increasing powers):

1. *eavesdropping*: can only eavesdrop the communication between the honest parties;
2. *input-/output-controlling*, *communication-halting*: can alter the input to the corrupted party, abort the execution, and modify the *local* output that is returned to the application context;
3. *state-controlling*: can compute the messages to be exchanged with the other party arbitrarily, but can only use the secure device in accordance to its interface (as prescribed by the protocol);
4. *active*: can break the tamper-resistance of the secure device, and thus can exert total control over the corrupted party during the execution of the protocol.

The above classification provides a spectrum of “partial” insider attacks, progressively bridging the gap between outsider and insider adversaries<sup>2</sup>. Besides we prove that output-privacy is attainable against state-controlling adversaries, thus tightening the gap between feasibility and impossibility results.

We note that theoretical modeling of protected and tamper-proof storage is a recent area of research, motivated by the advances in hardware technology and computer architectures. Reliance on some form of tamper-resistant hardware has been formally investigated in recent work, *e.g.*, the use of self-destructing capabilities for algorithmic tamper-proof security [8], of physical envelopes for collusion-free protocols [13, 14, 15].

<sup>1</sup> it closely resembles the *augmented semi-honest* model (*cf. e.g.*, [10, Chap. 7]).

<sup>2</sup> We assume that each secure devices embeds some randomness (*e.g.*, cryptographic keys) which is certified by a common PKI. If the operations were based on uncertified randomness, a state-controlling adversary in control of the driver could just pick fresh randomness and (perfectly) simulate the execution of the secure device, without being detected.

## 2.2 The Ideal Model

As in [3], the ideal process is parameterized by the functionality  $f : \mathbb{N} \times (\{0, 1\}^*)^2 \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times (\{0, 1\}^*)^2$  that the parties wish to evaluate and it assumes the existence of an incorruptible probabilistic trusted third party  $\mathcal{T}$ . Each party  $P_i$  is a pair of “dummy” Turing machines: a driver  $D_i$  (equipped with *input* and *local output* tapes) and a secure device  $SD_i$  (equipped with a *global output* tape). The secure device also exports a **Finalize**-query (that makes it write to its tape). An ideal-process adversary  $\mathcal{S}$  is an interactive PPT Turing machine which may influence the behavior of the controlled party.

In what follows, we describe the execution of the ideal process assuming an *input/output-controlling*, *communication-halting* adversary controlling one of the two party. The definition remains identical with *state-controlling* adversaries—in the ideal model, there is no communication, and can apply applies to the *active* scenario, since here the tamper-resistance of the secure device ought to be ideal.

### Input Stage

*Inputs to the parties:* Each  $P_i$  obtains an input  $x_i$ ; the controlled party communicates it to the adversary.

*Inputs to the trusted party:* An honest  $P_i$  always sends  $x_i$  to the trusted party  $\mathcal{T}$ . A controlled party  $P_{\bar{i}}$  may send some  $\hat{x}_{\bar{i}} \in \{0, 1\}^{|x_i|}$  to  $\mathcal{T}$ .

### Actual Computation

*$\mathcal{T}$  provides the local output to the first party:*  $\mathcal{T}$  evaluates  $(y_0, y_1, y_2) = f(x_1, x_2)$  inputs and sends  $y_1$  to  $P_1$ ’s driver.

*$\mathcal{T}$  provides the local output to the second party:* If  $P_1$  is controlled, it hands off  $y_1$  to  $\mathcal{S}$ . Then  $\mathcal{S}$  may decide to “stop”  $\mathcal{T}$ . In this case,  $\mathcal{T}$  sends  $\perp$  to  $P_2$ ’s driver and to both parties’ secure devices. Otherwise,  $\mathcal{T}$  sends  $y_2$  to  $P_2$ ’s driver.

*$\mathcal{T}$  provides the global output to the first party:* If  $P_2$  is controlled, it hands off  $y_2$  to  $\mathcal{S}$ . Then  $\mathcal{S}$  may decide to “stop”  $\mathcal{T}$ . In this case,  $\mathcal{T}$  sends  $\perp$  to both parties’ secure devices. Otherwise,  $\mathcal{T}$  sends  $y_0$  to  $SD_1$ .

*$\mathcal{T}$  provides the global output to the second party:* If  $P_1$  is controlled, the adversary may decide to “stop”  $\mathcal{T}$ . In this case,  $\mathcal{T}$  sends  $\perp$  to  $SD_2$  else it sends  $y_0$  to  $SD_2$ .

### Output Stage

*Local output:* An honest  $P_i$  writes his  $y_i$  to its local output tape, while a controlled party’s driver may write an arbitrary value.

*Global output:* An honest  $P_i$  invokes the **Finalize**-query on  $SD_i$ , which causes it to write  $y_0$  on its global output tape. A controlled party’s driver may or may not invoke the **Finalize**-query. If such query is not invoked, the global output tape contains  $\perp$ , otherwise the secure device writes  $y_0$  to its global output tape.

## 3 Main Result

The first theorem shows that our definition formally captures the intuition that private-output multiparty computation should be impossible when facing active adversaries. The second theorem shows that still interesting constructions are possible.

**Theorem 1.** *Let  $f$  be any 1-non-trivial two-party function (with respect to some distribution  $\mathcal{D}$ ). There exists an active adversary  $\mathcal{A}$  such that no protocol  $\pi$  can output-privately compute  $f$  in the presence of  $\mathcal{A}$ .*

**Theorem 2.** *If enhanced trapdoor permutations exist, then for any private-output two-party functionality  $f$ , there exists a protocol  $\pi$  output-privately implementing  $f$  with respect to STATE-adversaries.*

In a nutshell our proof first describes an adequate hybrid model, then shows that this model can be used to *reduce* the problem to that of realizing deterministic functionalities and finally, we exhibit a construction for any deterministic function seen as an arithmetic circuit. As a by-product we obtain the following composition theorem.

**Theorem 3 (informal).** *Assume a protocol  $\Pi^g$  output-privately computes a functionality  $g$ , and a protocol  $\Pi^{f|g}$  output-privately computes  $f$  using ideal calls to  $g$ . Then the composed protocol  $\Pi^f \doteq \Pi^{f|g} \circ \Pi^g$  output-privately computes  $f$  in the real model.*

## References

1. R. Bar-Yehuda, B. Chor, E. Kushilevitz, and A. Orlitsky. Privacy, Additional Information, and Communication. *IEEE IT* 39(6):1930-1943, 1993.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *STOC '88*, pp. 1–10. ACM.
3. R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *J. of Cryptology*, 13(1):143–202, 2000.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *FOCS '01*, pp. 136–145. Eprint 2001/067.
5. R. Canetti, U. Feige, O. Goldreich and M. Naor Adaptively Secure Computation *STOC '96*, pp. 639–648. ACM.
6. D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols. *STOC '88*, pp.11–19. ACM.
7. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. on Discrete Mathematics*, 4(1):36–47, 1991. Earlier version in *STOC '89*, pp. 62–72. ACM.
8. R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali and T. Rabin. Algorithmic Tamper-Proof Security: Theoretical Foundations for Security against Hardware Tampering. *TCC '04*.
9. R. Gennaro, M. Rabin and T. Rabin Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography. *PODC '98*, pp. 101–111. ACM.
10. O. Goldreich. Foundations of cryptography, Volume 2. Basic Applications. 2004.
11. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. *STOC '87*, pp. 218–229. ACM.
12. S. Halevi, D. Naor and P. Karger. Enforcing Confinement in Distributed Storage and a Cryptographic Model for Access Control <http://eprint.iacr.org/2005/169/>.
13. M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely Fair SFE and coalition-safe cheap talk. *PODC '04*, pp. 1-10. ACM.
14. M. Lepinski, S. Micali, and A. Shelat. Fair Zero Knowledge. *TCC 05*, pp. 245-263. Springer.
15. M. Lepinski, S. Micali, and A. Shelat. Collusion-free Protocols. *STOC '05*, pp. 543-552.
16. A. Yao. Protocols for Secure Computations. *FOCS '82*, pp. 160-164. IEEE.