

Output-Privacy in Secure Multiparty Computation

Emmanuel Bresson Dario Catalano Nelly Fazio
Antonio Nicolosi Moti Yung

CELAR, France

ENS, France

NY University, USA

Columbia University & RSA Labs, USA

June 20, 2006

- 1 Introduction
 - Secure Multi-Party Computation
 - Our Motivation
 - Difficulties
- 2 Output Privacy: a Definitional Effort
 - Main Ideas
 - The Real Model
 - Modelling Adversaries
 - The Ideal Model
- 3 Main Results
 - Impossibility Result
 - Feasibility Result
- 4 Conclusion

Our results

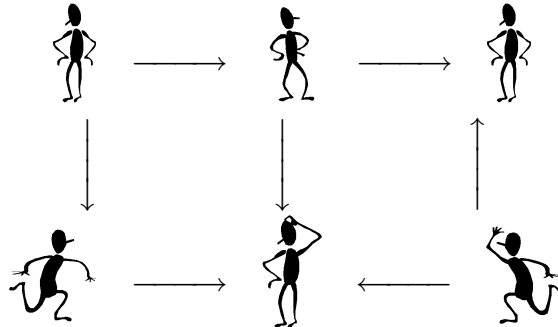
- define *private-output multiparty computation*
- encompass the classical definition as a particular case
- give both impossibility and feasibility results.

- 1 Introduction
 - Secure Multi-Party Computation
 - Our Motivation
 - Difficulties
- 2 Output Privacy: a Definitional Effort
- 3 Main Results
- 4 Conclusion

Multi-Party Computation

Introduced in [Yao82, GMW87]

Enables a set of parties to jointly compute a function, while keeping their inputs private.



Secure Multi-Party Computation

Security is normally formulated in a simulation paradigm [Bea91, Can00]:

Real world: parties carry out the protocol, adversary \mathcal{A} controls communication and corrupts parties;

Ideal world: a functionality \mathcal{F} does the computation, ideal adversary \mathcal{S} is rather limited;

Security: protocol Π securely realizes \mathcal{F} , if

$$\forall \mathcal{A}, \exists \mathcal{S} \text{ s.t. } \text{View}_{\mathcal{A}}(\Pi) \approx \text{View}_{\mathcal{S}}(\mathcal{F})$$

The power of the framework

According to various settings (synchronicity, channel privacy, concurrency, reactivity, corruption, cryptographic assumptions),
it is possible to compile any poly-sized function into a protocol that maintains input privacy.

Major results known: secure MPC exists. . .

. . . in the computational model [GMW87]:

- against passive adversary and any number of dishonest,
- against active adversary corrupting dishonest minority;

. . . in the information-theory setting [CCD88, BGW88]:

- against passive adversary and dishonest minority,
- against active adversary and up to $1/3$ dishonest parties.

Limitations of the framework

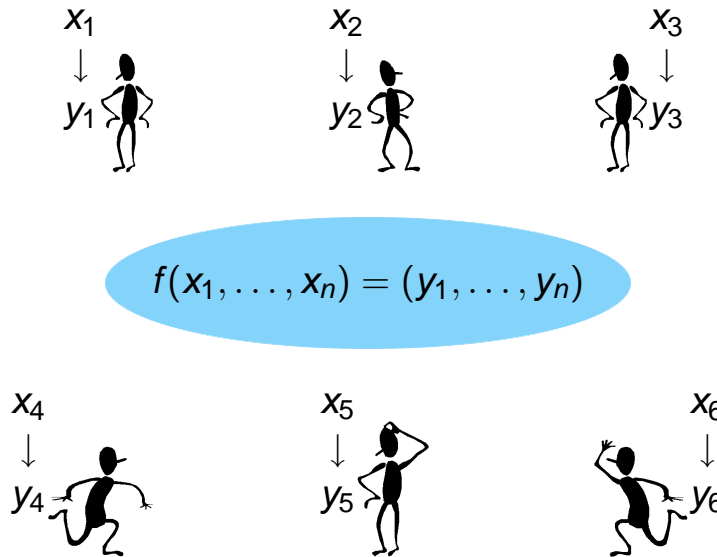
The privacy concern [ChKu89, BCKO93] is about the honest parties' *inputs*: \mathcal{A} controls the bad parties' inputs, the function's output is public.

What about confidentiality of the *output(s)*?

Leakage of outputs might have catastrophic consequences for the security:

- if the computed output is a secret key;
- if there is a need to reveal the output only at a certain point, in "real time," and not before.

Multi-Party Computation



What happens if some (or all) y_i are the same?

Output-Privacy: a concern

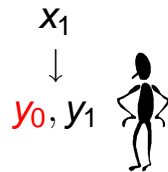
We concentrate on functionalities for which there is a *global output* y_0 common to all players.

*Moreover, protocol's **correctness** should guarantee that it is actually the case.*

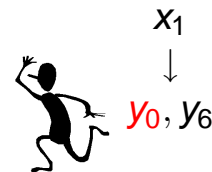
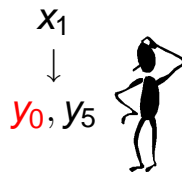
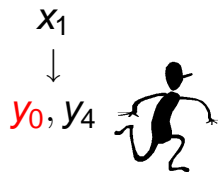
Question

Can we do better than the classical case and guarantee privacy for the global output, even when some corruptions occur?

MPC with Common Output



$$f(x_1, \dots, x_n) = (y_0, y_1, \dots, y_n)$$



A single corruption suffices to reveal all outputs

Intuitive achievements

What should be...

- 1 Output-privacy with respect to traditional *active adversaries* seems impossible
- 2 Security against “outsiders” should be achievable...
- 3 ... but is of low interest.

Question

- What is the highest attainable level of protection?
- What about “intermediate” attackers?

⇒ Find the right definition

Corruption by active adversaries, revisited

When there exists a common output to both parties, we (naturally) 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)

Usual adversarial settings (*eavesdropping, honest-but-curious, active*) form a too restrictive classification for output-privacy.

Change the laws of corruption!

- 1 Introduction
- 2 Output Privacy: a Definitional Effort
 - Main Ideas
 - The Real Model
 - Modelling Adversaries
 - The Ideal Model
- 3 Main Results
- 4 Conclusion

A simple observation

How the output is computed

Each player computes the global output using:

- its input,
- its random coins,
- the messages exchanged with the other parties.

On corruption, a “classical” adversary gets all these three quantities.

Our approach

The adversary’s power is restricted by postulating that it cannot access (part of) these three quantities, *even for the corrupted players*.

- Communication is public \implies known to the adversary
- Input may be decided by a higher-level protocol \implies likely to be known
- Restrict the adversary’s access to the controlled parties’ randomness and outputs.

New classes of adversaries

We classify the “intermediate” adversaries depending on how and when they can access to this randomness.

Adversaries of increasing capabilities (the more randomness \mathcal{A} is allowed to access, the more powerful it is):

- 1 All randomness accessible == the classical definition (we implicitly set $y_0 = \perp$)
- 2 No randomness accessible: a fully private output ($y_i = \perp$)

The real model

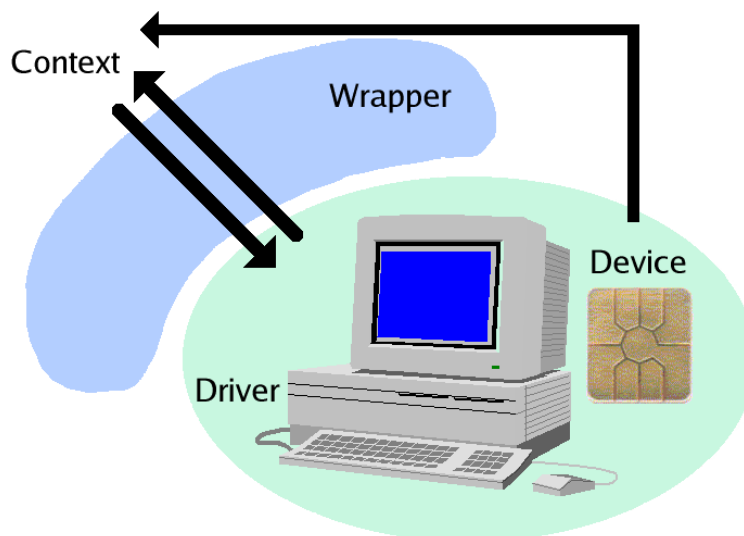
We split protocol’s participants in several components:

- helps in discussing local outputs/global output
- helps in separating accessible from hidden randomness
- helps in classifying adversaries
- modular implementation?

Component-wise overview

- 1 *application context*: provides input, obtains outputs,
- 2 *control wrapper*: 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 party's code: each is split into
 - the *driver* (in charge of high-level operations)
 - the *secure device*, which has *limited* capabilities, but carries out only the most sensitive computations.

Component-wise overview



Driver and wrapper never see the global output!

Motivation for using secure devices

Motivation

Theoretical modelling of protected and tamper-proof storage is a recent area of research:

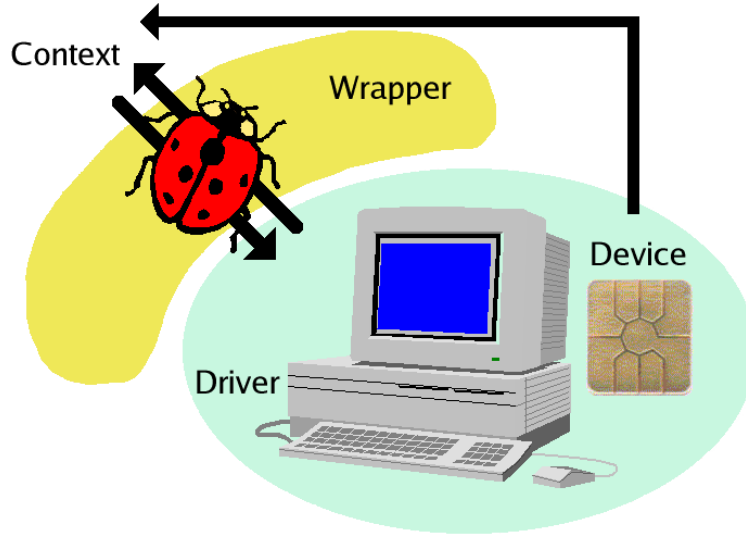
- advances in hardware technology and computer architectures,
- tamper-resistant hardware has been recently investigated:
 - use of self-destructing capabilities for algorithmic tamper-proof security [GLM+04],
 - physical envelopes for collusion-free protocols [LMPS04, LMS05, LMS05b].

Different attacks

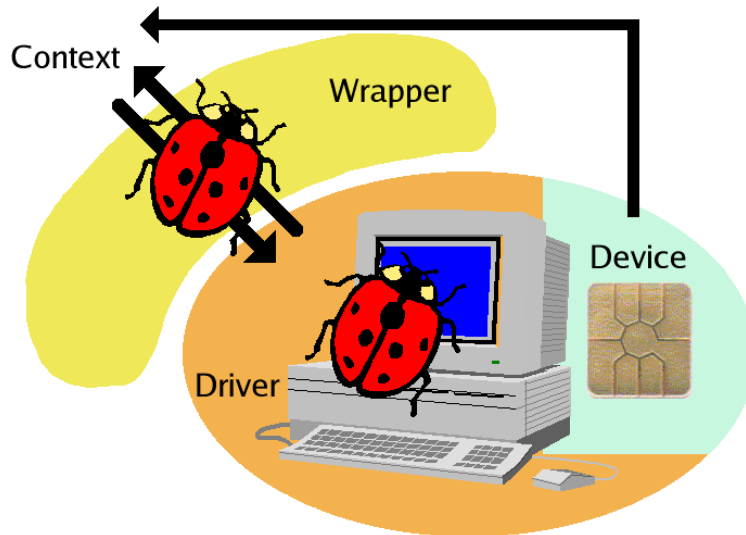
The 4 kinds of adversaries

- 1 *eavesdropping* = eavesdrop the communication between the honest parties;
- 2 *input-/output-controlling, communication-halting* = subvert the control wrapper mechanism;
- 3 *state-controlling* = replace the code for the driver itself. Still restricted to use the interface provided by the tamper-resistant device;
- 4 *active* = obtain full control over the corrupted party.

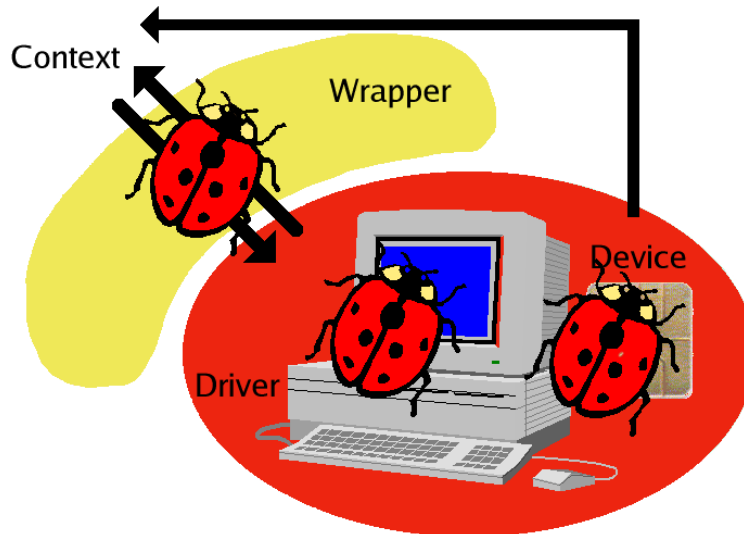
Input/output-controlling, halting adversary



State-controlling adversary



Active adversary



Security is impossible here...

Remarks about secure devices

To meaningfully separate state-controlling adversaries from active ones, we assume each secure device to embed some randomness (e.g., cryptographic keys) which is certified by a common PKI.

With operations based on uncertified randomness, a *state-controlling* adversary could just pick fresh randomness and (perfectly) simulate the execution of the secure device, without being detected.

The secure device should not be specific to any functionality: its interface should be *generic* and as small as possible.

Technical details

- The *partial* view of player P_i :

$$\text{VIEW}_i^P(x_1, x_2) = (x_i, r_i^{\text{Driver}}, m_1, \dots, m_t, \text{SD-queries}),$$

the local output $\text{OUT}_i^L(x_1, x_2)$ can be derived from it;

- the *complete* view of player P_i :

$$\text{VIEW}_i^C(x_1, x_2) = (x_i, r_i, m_1, \dots, m_t),$$

the global output $\text{OUT}_i^G(x_1, x_2)$ can be derived from it;

Definition (Protocol view in the real world)

The view of the protocol Π in the real world is defined as

$$\text{VIEW}^\Pi(x_1, x_2) = \left(\text{OUT}^G(x_1, x_2), \text{OUT}_1^L(x_1, x_2), \text{OUT}_2^L(x_1, x_2) \right)_{r_i},$$

with $\text{OUT}^G(x_1, x_2) \neq \perp$ iff $\text{OUT}_1^G(x_1, x_2) = \text{OUT}_2^G(x_1, x_2)$.

Technical details (cont.)

In the presence of \mathcal{A} :

- if \mathcal{A} is passive, he gets the (m_1, \dots, m_t) ;
- if \mathcal{A} is I/O-controlling, he gets $\text{VIEW}_i^P(x_1, x_2) = (x_i, r_i^{\text{Driver}}, m_1, \dots, m_t, \text{SD-queries})$, he can change the m_i 's and may choose to "halt";
- if \mathcal{A} is state-controlling, he can compute the m_i himself, but has to use SD_i interface;
- if \mathcal{A} is active, he gets $\text{VIEW}_i^C(x_1, x_2) = (x_i, r_i, m_1, \dots, m_t)$, and so learns y_0 .

Definition (Protocol in the presence of \mathcal{A})

$$\text{REAL}_{\mathcal{A}}^\Pi(x_1, x_2) = \left((\text{OUT}_1^G(\cdot, \cdot), \widehat{\text{OUT}}_1^L(\cdot, \cdot)), (\text{OUT}_2^G(\cdot, \cdot), \widehat{\text{OUT}}_2^L(\cdot, \cdot)) \right)_{r_{\mathcal{A}}, r_i}$$

The ideal model

- Parameterized by the functionality

$$f : \mathbb{N} \times (\{0, 1\}^*)^2 \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times (\{0, 1\}^*)^2$$
- assume an incorruptible probabilistic trusted third party \mathcal{T} .
- each party P_i is a pair of “dummy” Turing machines
 - a driver D_i , with *input* and *local output* tapes
 - a perfectly secure device SD_i with a *global output* tape

The secure device also exports a Finalize-query (that makes it write to its tape).
- adversary \mathcal{S} = interactive PPT Turing machine

Description of the computation

Input stage:

- P_i obtains x_i ; if corrupted, gives it to \mathcal{A}
- if honest, gives x_i to \mathcal{T} , else may give some \hat{x}_i

Actual computation:

- \mathcal{T} sets $(y_0, y_1, y_2) = f(x_1, x_2)$ and sends y_1 to D_1
- If P_1 corrupted, gives y_1 to \mathcal{S} , which may “halt”. If not, \mathcal{T} sends y_2 to D_2
- If P_2 corrupted, gives y_2 to \mathcal{S} , which may “halt”. If not, \mathcal{T} sends y_0 to SD_1
- If P_1 corrupted, \mathcal{S} may “halt”. If not, \mathcal{T} sends y_0 to SD_2

Output stage:

- honest P_i writes his y_i to its local output, else writes anything
- honest P_i invokes the Finalize-query on SD_i

Ideal adversaries' view

- On input $x_{\bar{i}}$, \mathcal{S} may change it to $\hat{x}_{\bar{i}}$; $P_{\bar{i}}$ corrupted
- When receiving $y_{\bar{i}}$, \mathcal{S} computes $\hat{y}_{\bar{i}}$ and $(t_{\text{HALT}}, b_{\text{FIN}}) \in \{\text{EARLY, LATE, NEVER}\} \times \{\text{YES, NO}\}$;
 - t_{HALT} specifies when to abort (if ever),
 - b_{FIN} specifies if $SD_{\bar{i}}$ is allowed to output y_0 .

Definition (Ideal world execution)

$\text{IDEAL}_{\mathcal{S}}^f(x_1, x_2) = ((y_0^1, \hat{y}_1), (y_0^2, \hat{y}_2))$ with:

- $\hat{x}_{\bar{i}}$ set by \mathcal{S} and $\hat{x}_{3-\bar{i}} = x_{3-\bar{i}}$,
- $(y_0, y_1, y_2) \leftarrow f(\hat{x}_1, \hat{x}_2)$,
- $\hat{y}_{\bar{i}}$ set by \mathcal{S} and $\hat{y}_{3-\bar{i}} = y_{3-\bar{i}}$ unless EARLY halting, $y_0^{\bar{i}} = y_0$ unless NO output, $y_0^{3-\bar{i}} = y_0$ only if NEVER halting.

Security definition

Definition

Let f a global-output functionality. A protocol Π is said to output-privately compute f if for any real world adversary \mathcal{A} , there exists an ideal adversary \mathcal{S} (with corresponding capabilities of corruption) such that the following distribution ensembles are computationally indistinguishable:

$$\{\text{IDEAL}_{\mathcal{S}}^f(x_1, x_2)\}_{x_1, x_2} \stackrel{c}{\approx} \{\text{REAL}_{\mathcal{A}}^{\Pi}(x_1, x_2)\}_{x_1, x_2}$$

- 1 Introduction
- 2 Output Privacy: a Definitional Effort
- 3 Main Results
 - Impossibility Result
 - Feasibility Result
- 4 Conclusion

Impossibility result

Theorem

Let f a functionality whose global output depends on “both” inputs. There exists an active adversary \mathcal{A} such that no protocol π can output-privately compute f in the presence of \mathcal{A} .

Intuition... (assume P_1 corrupted)

An active real adversary \mathcal{A} gets the value of y_0^1 and outputs it as part of $\widehat{\text{OUT}}_1^L(\cdot, \cdot)$ in $\text{REAL}_{\mathcal{A}}^{\Pi}(x_1, x_2)$

While for any ideal adversary \mathcal{S} that sees only (x_1, y_1) cannot infer about y_0 (due to the requirement on f).

Then $\text{IDEAL}_{\mathcal{S}}^f(x_1, x_2) \not\approx \text{REAL}_{\mathcal{A}}^{\Pi}(x_1, x_2)$.

Feasibility result

Theorem

If enhanced trapdoor permutations exist, then for any private-output two-party functionality f , there exists a protocol π output-privately implementing f with respect to STATE- \mathcal{A} .

Intuition . . .

Define an hybrid model with “ideal-calls” to a functionality: such call is answered with local outputs being given to drivers, and global outputs being (securely) given to devices.

Show that standard notion of composability still applies.

Reduce the computation of probabilistic functionalities to that of deterministic ones, in the appropriate hybrid model.





Construct a protocol for deterministic functionalities.

Conclusion and further work





Work still under progress . . .

- Enhance the composability framework to a “UC private output” one
- Refine the classification of adversaries
- Find new applications (agreement, contract signing, . . .)
- Describe practical implementations





Bibliography I

-  [R. Bar-Yehuda, B. Chor, E. Kushilevitz, and A. Orlitsky.](#)
Privacy, Additional Information, and Communication.
IEEE IT 39(6):1930-1943, 1993.
-  [D. Beaver.](#)
Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority.
J. of Cryptology, 4(2):75–122, 1991.
-  [M. Ben-Or, S. Goldwasser, and A. Wigderson.](#)
Completeness theorems for non-cryptographic fault-tolerant distributed computation.
STOC '88, pp. 1–10. ACM.
-  [R. Canetti.](#)
Security and Composition of Multi-party Cryptographic Protocols.
J. of Cryptology, 13(1):143–202, 2000.


Bibliography II

-  [D. Chaum, C. Crépeau, and I. Damgård.](#)
Multiparty Unconditionally Secure Protocols.
STOC '88, pp.11–19. ACM.
-  [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.
-  [R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali and T. Rabin.](#)
Algorithmic Tamper-Proof Security: Theoretical Foundations for Security against Hardware Tampering.
TCC '04. pp. 258–277.
-  [O. Goldreich.](#)
Foundations of cryptography, Volume 2. Basic Applications.
2004.

Bibliography III

-  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.
-  M. Lepinski, S. Micali, C. Peikert, and A. Shelat.
Completely Fair SFE and coalition-safe cheap talk.
PODC '04, pp. 1-10. ACM.
-  M. Lepinski, S. Micali, and A. Shelat.
Fair Zero Knowledge.
TCC 05, pp. 245-263. Springer.
-  M. Lepinski, S. Micali, and A. Shelat.
Collusion-free Protocols.
STOC '05, pp. 543-552. ACM.

Bibliography IV

-  A. Yao.
Protocols for Secure Computations.
FOCS '82, pp. 160-164. IEEE.