Incoercible Multiparty Computation

(Extended Abstract)

Ran Canetti* Rosario Gennaro*

May 17, 1996

Abstract

Current secure multiparty protocols have the following deficiency. The public transcript of the communication can be used as an involuntary *commitment* of the parties to their inputs and outputs. Thus parties can be later coerced by some authority to reveal their private data. Previous work that has pointed this interesting problem out contained only partial treatment.

In this work we present the first general and rigorous treatment of the coercion problem in secure computation. First we present a general definition of protocols that provide resilience to coercion. Our definition constitutes a natural extension of the general paradigm used for defining secure multiparty protocols. Next we show that if trapdoor permutations exist then any function can be incoercibly computed (i.e., computed by a protocol that provides resilience to coercion) in the presence of computationally bounded adversaries and only public communication channels. This holds as long as less than half the parties are coerced (or corrupted). In particular, ours are the first incoercible protocols without physical assumptions. Also, our protocols constitute an alternative solution to the recently solved adaptive security problem.

Our techniques are quite surprising and include non-standard use of deniable encryptions.

*Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge MA 02139, U.S.A. canetti,rosario@theory.lcs.mit.edu

1 Introduction

Consider a set of parties who do not trust each other, nor the privacy of the channels by which they communicate. Still, the parties want to compute some common function of their inputs in a secure way. Security here means maintaining correctness of the outputs while keeping the parties' internal data as private as possible. This is the well-known secure multiparty computation problem (e.g., [Y, GMW2]). The parties' distrust in each other is modeled via an adversary that corrupts parties, learns their inputs, and controls their behavior. Distrust in the communication channels is modeled by allowing the adversary to 'listen in' on all the communication. (We assume that the adversary cannot *modify* the communication between uncorrupted parties.) Limiting the adversary to probabilistic polynomial time, we call this the computational setting for multiparty computation.

Protocols for securely computing any function in this and related settings have been known for a while ([GMW2, BGW, CCD, RB, CFGN] and many others). These protocols are quite strong. As long as enough parties remain uncorrupted, the parties can correctly compute any function of their inputs, while making sure that the adversary learns nothing from the computation other than the inputs and outputs of the corrupted parties. However, these protocols suffer from the following deficiency, pointed out in [He, BT]. Using any of these protocols, the public transcript of the communication can be used as an (involuntary) *commitment* of the parties to their inputs and outputs. That is, consider a party that is required (say, by some coercive agent) to present its input and output of a computation that has just been completed (or may still be in execution), as well as all random choices made during the computation. The presented values are then matched by the coercer against the known transcript, checking for mismatches. Using known protocols, it is infeasible (or even impossible) for the coerced party to find inputs, outputs, and random choices, *other than the real ones,* that match the public transcript (or, more generally, match the information already known to the coercer). Interestingly, a similar phenomenon happens even when the communication channels are private; we elaborate in the sequel. Getting rid of this seemingly inherent deficiency of secure protocols is the focus of this work.

A classic scenario where the 'coercion problem' is acute is secret voting, where vote-buying may result. Other scenarios include electronic commerce, bidding, key-escrowing, and any computation done in the presence of some authority (say, a Mafia, an employer, or a government) that may become coercive.

Coercion in the context of secret voting has been studied in the past [BT, SK, NR]. These studies, however, were limited. First, they consider only a simplified version of secret voting, where there are voting centers that cannot be coerced. Next, their constructions require physically secure communication channels during crucial parts of the computation. But most importantly, they define incoercible ('receipt-free', in their language) voting protocols only in a limited way. In particular, their definitions do not seem to apply to general secure computation (nor to the problem of incoercible voting in its full generality).

This work initiates a study of resilience to coercion in general multiparty secure computation. First we suggest a general, rigorous definition of what it means for a protocol to provide resilience to coercion. Coming up with a definition that correctly captures this new type of attack turned out to be a non-trivial task. Moreover, our definition manages to incorporate this concern within the existing framework used for defining multiparty secure computation [MR, Be, CFGN, C]. In particular, it is immediate from our definition that any incoercible protocol is also (adaptively) secure in the standard sense. Next we present a construction for securely and incoercibly computing any function in the computational setting, as long as less than half the parties are being coerced (or corrupted). In the rest of the introduction we discuss our definition and construction in more detail.

DEFINING INCOERCIBLE COMPUTATION. In a nutshell, we define incoercibility as the ability to present the coercer with 'fake random input' that will make the public transcript 'look as if' it originated with input different than the one really used. Let us elaborate.

We allow a coercive agent to coerce parties to reveal their internal data. (Coercion may take place at any time during or after the computation). For simplicity we unite this coercive agent with the 'standard' adversary used for defining security. That is, the adversary can now choose to either **corrupt** or **coerce** a party. Unlike corrupted parties, coerced parties are not obliged to present the coercer with their *real* internal data. Instead, a coerced party presents the coercer with data that may be either its real internal data or fake. This is modeled as follows. First we let each party have, on top of its input x (to be used in the computation), also a **fake input** x', to be presented upon coercion. (x' is arbitrary, and can be thought of as decided upon at the time of coercion.) When the adversary *corrupts* a party it is given x, together with the party's (real) random input r. (All other internal data can be reconstructed from x, r, and the public communication.) When the adversary *coerces* a party, it is given x' together with a **fake random input** r', where r' is computed as follows. If $x = x'$ then $r' = r$. Else, r' is generated via a special **faking algorithm** run locally by the party. This faking algorithm is public and specified in the protocol. (An additional concern is the ability to protect also the party's *private output* from coercion. We treat this concern in a similar way.)

In principle, we would want to say that a protocol provides good protection from coercion if, upon coercion, the adversary is "unable to detect whether $x' = x$ " (that is, whether or not it was given the party's real input). But there are cases where it is obvious that $x' \neq x$, thus detection is unavoidable. (For instance, assume that the computed function defines B's output to be equal to A's input, but corrupted B presents output 1 and coerced A presents $x' = 0$.)

So we want to capture the cases where detection is unavoidable, and say that a protocol provides good protection from coercion if it allows only unavoidable detection. We do this using standard methodology. We first envision an **ideal model** of computation, which captures 'the most we can hope for' from *incoercible* computation. We then say that a protocol (that now includes also instructions for the case of coercion) for computing a function is **incoercible** if running it is equivalent to computing the function in the ideal model. This ideal model, described in the sequel, extends of the ideal model used to define 'standard' security of multiparty protocols [MR, Be, CFGN, C].

Having formalized the notion of incoercible protocols, a simple argument now shows that incoercible secure computation is impossible when the adversary is computationally unbounded. Interestingly, this holds even if the communication channels are private (as in [BGW, CCD]). Thus, our protocols are another demonstration of the interesting fact that constructions based on the computational limitations of the adversary can achieve qualitatively stronger results than constructions based on physical assumptions (such as private channels).

ON THE CONSTRUCTION. We construct incoercible protocols for computing any function in the computational setting, as long as less than half of the parties are coerced or corrupted. A main tool in our construction is a new type of encryption schemes, called **deniable encryptions** [CDNO]. These schemes address the same problem as here, in the limited context of encryption. That is, on top of standard (semantic) security, in a deniable encryption scheme the sender of the ciphertext (or, alternatively, the receiver) is able to present a coercer with a **fake random input** that is consistent with the original ciphertext and any arbitrary cleartext. Consequently, coercion does not help in finding the transmitted value.

A natural first attempt at constructing incoercible protocols may be to start with any standard secure protocol (say, [GMW2, CFGN]) and have the parties encrypt each message using deniable encryption. However, this simple solution does not work. Essentially, the reason is that each encrypted message (i.e., each ciphertext) has a unique receiver who can correctly decrypt it. Thus whatever the sender claims regarding the cleartext can be matched by the adversary against the value decrypted by the receiver (in case the receiver is corrupted), or against the receiver's 'story' (in case the receiver is coerced). Consequently, any attempt to present fake input different than the real one will be detected by the adversary with high probability.

We solve this problem by using deniable encryptions *with no specified receiver.* Instead, the ciphertext will be broadcasted and the decryption key will be shared among the players using a threshold sharing scheme. This will allow the players to *jointly* use the information contained in the encrypted messages, but will prevent each individual party (or a small set of parties) from decrypting — thus making their coercion useless.

Unfortunately, current constructions of deniable encryption fail to reduce the probability of successful coercion to negligible values. Instead, the probability of successful coercion, or the **coercion probability**, decreases only polynomially fast in the security parameter (see details in the sequel). This weakness propagates to our construction; however, given a deniable encryption scheme with negligible coercion probability, our construction also have negligible coercion probability. It is an interesting open problem to construct deniable encryption schemes with negligible coercion probability.

On incoercibility and adaptive security. Recently it was shown how to compute any function in the computational setting when the adversary is **adaptive** (i.e., when it can corrupt parties during the course of the computation based on the information gathered so far) [CFGN]. In principle, adaptive security (i.e., security against adaptive adversaries) is unrelated to incoercibility. In particular, the coercion problem remains even if the set of coerced parties is known in advance. Nevertheless, our solution achieves both incoercibility and adaptive security.

For convenience, we use the [CFGN] protocol in our construction. However, we can do with protocols that achieve only a weaker security property, described in the sequel (the [GMW2] protocols, as well as the [BGW] protocols augmented with encrypting each message using standard encryption, have this property). In fact, our construction is an alternative to [CFGN] for obtaining adaptive security. (A third alternative is proposed in [CDNO].)

We remark that [CFGN] construct yet another type of encryption functions, called **non-committing encryption**. Non-committing encryptions have similar flavor to deniable ones, in that there exist ciphertexts that can be "opened" as encryptions of, say, both 1 and 0. However, non-committing encryptions are *weaker* than deniable ones. (One main difference, among others, is that in non-committing encryptions the parties *using* the scheme are, in general, unable to generate ciphertexts that can be opened both ways. Such ciphertexts can only be generated in a simulated execution of the protocol (such as the one needed there). In deniable encryption each ciphertext generated by parties has unique decryption, and at the same time can be "opened" both ways for a coercer.)

Organization. Section 2 is dedicated to defining incoercible protocols. Next, in Section 3 we present a simple argument demonstrating that incoercibility cannot be achieved in the presence of unbounded adversaries. The same argument demonstrates why existing constructions are coercible, even when the adversary is limited to probabilistic polynomial time. In Section 4 we describe two tool used in our construction: deniable encryption and adaptively secure protocols. In Section 5 we describe and analyze our constructions; Finally, in Section 6 we mention some open problems.

2 Definitions

We propose a definition of incoercible multiparty computation, that regards incoercibility as an extension of standard (adaptive) security. We first present a basic definition, then in Section 2.2 we extend the definition to a more general case. In Section 2.3 we discuss some properties of the definition.

Let us first recall the definition of indistinguishability of distributions. We present a slightly more general definition than the standard one, accounting also for distances that are more than negligible. (A real-valued function is called **negligible** if it approaches zero faster than any polynomial.)

Definition 1 *Let $\mathcal{A} = \{A_n\}_{n\in\mathbf{N}}$ and $\mathcal{B} = \{B_n\}_{n\in\mathbf{N}}$ be two ensembles of probability distributions, and let $\delta : \mathbf{N} \to [0,1]$. We say that $\mathcal{A}$ and $\mathcal{B}$ are $\delta(n)$-***computationally close** *if for every polytime distinguisher* Dist *and for all large enough n, $|\mathrm{Prob}(\mathsf{Dist}(A_n) = 1) - \mathrm{Prob}(\mathsf{Dist}(B_n) = 1)| < \delta(n)$.*

If $\delta(n)$ is negligible then we say that $\{A_n\}$ and $\{B_n\}$ are computationally indistinguishable *and write $A_n \stackrel{c}{\approx} B_n$.*

2.1 The basic definition

Our definition builds on top of the existing framework used for defining multiparty computation secure against adaptive adversaries ([C, CFGN], following [MR, Be]). We proceed in three steps: First we describe the 'mechanics' of a computation in the presence of a coercive adversary, emphasizing the difference from the 'mechanics' of a computation in the presence of a standard adaptive adversary. Next we define an ideal model of computation, which represents the 'most we can hope for' from incoercible computation. This ideal model is an extension of the ideal model used in [C, CFGN] to define adaptive security. Finally we say that a (real-life) computation is incoercible if it is equivalent — in a standard sense described in the sequel — to a computation in the ideal model. Having used this framework, our definition treats incoercibility as a natural extension of standard (adaptive) security. In particular, it is obvious from the definition that any incoercible protocol is also secure in the standard sense. We believe that this is an important feature of the definition.

In the rest of this subsection we describe the definition in detail. The details turn out to be quite important here, as there are several subtle issues that need to be considered. We have tried to point these issues out in the presentation below and in Section 2.3.

THE REAL-LIFE COMPUTATION. We first describe the standard computational setting for secure multiparty computation [GMW2]. Next we incorporate coercion in this setting. There are n parties, each with its private input. (Each party, as well as the *adversary* and the *ideal-model-adversary* postulated in the sequel, is an interactive Turing machine limited to probabilistic polynomial time.[1]) The parties wish to compute a common function f of their inputs. For now we assume that all parties have one, common output. The more general case where each party may have its own *private* output requires somewhat different treatment and is dealt with in Section 2.2. Denote the parties by $P_1, \ldots, P_n$, where each P_i has private input x_i and sufficiently long random input r_i.

The parties engage in a distributed computation, following a protocol π, in order to evaluate f at their inputs.[2] The system is synchronous. Also for simplicity we assume that all communication is done on a broadcast channel. That is, each message sent by some party at some communication round is received in full by all other parties before the beginning of the next round. An adaptive adversary sees all the communication between parties and may corrupt parties during the course of the computation, based on the information seen so far. Once a party is corrupted, the adversary sees the party's (real) internal data. (Without loss of generality we assume that this data consists only of the party's input and random input. All the other internal data can be computed from the input, random input, and the messages received by the party so far.) From this round on, the corrupted party follows the instructions of the adversary. In particular, the party outputs whatever is instructed by the adversary. Without loss of generality, we assume that this output is the adversary's entire view of the computation. (The adversary view of the computation consists of its random input and all the information seen during the computation. In the sequel we specify this information in more detail.)

COERCION. We think of coercion as a weaker form of corruption, in which the adversary does not have direct access to the coerced party's data. Instead, the party presents the adversary with data that could be either its real internal data, or fake. This is modeled as follows. In addition to its input x_i,

[1] For simplicity, we assume that the computed function is polynomially computable. Alternatively, the simulator is polynomial in the complexity of the function.

[2] We assume that protocol π does not instruct the parties to *erase* any data during the computation. Alternatively one may assume that the parties copy all data to a safe, non-erasable memory. See [CFGN] for a discussion on parties that avoid erasing data, and about semi-honest parties in general.

each party P_i has an arbitrary **fake input** x'_i. Also, in addition to corrupting parties, the adversary can now **coerce** parties during the course of the computation. (Henceforth we call the adversary a **coercive adversary**.) Upon coercion, party P_i presents the adversary with the fake input x'_i instead of the real input x_i.[3] Furthermore, P_i presents the adversary with a **claimed random input** r'_i as follows. If $x'_i = x_i$ then $r'_i = r_i$ where r_i is P_i's real random input. Otherwise, r'_i is computed via a special **faking algorithm**, ϕ, that is specified in the protocol and is publicly known. Algorithm ϕ takes four inputs: x_i, x'_i, r_i, and COM_i, where COM_i is all the messages received by P_i so far. That is, if $x_i \neq x'_i$ then $r'_i = \phi(x_i, x'_i, r_i, \text{COM}_i)$. We stress that ϕ is computed by P_i based on locally known information. Informally, the definition will imply that r'_i does not 'help' the adversary to distinguish between the case where $x_i = x'_i$ and the case where $x_i \neq x'_i$.[4]

We remark that this is a conceivably stronger adversary model than the model where the corrupting and coercing entities are separate and do not share information. We chose this model since it is conceptually simpler, and since our protocols withstand it.[5] [6] Say that a coercive adversary is (c,t)-limited if it coerces up to c parties and corrupts up to t (presumably, but not necessarily, different) parties during the computation.

We use the following notation. Let $\vec{x} = x_1, \ldots, x_n$, $\vec{x}' = x'_1, \ldots, x'_n$, and $\vec{r} = r_1, \ldots, r_n$. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}', \vec{r})$ denote the **view** of the adversary $\mathcal{A}$ when interacting with parties running protocol π on input $\vec{x}$, fake input $\vec{x}'$, and random input $\vec{r}$ (x_i, x'_i and r_i for party P_i). The view contains the adversary's random input and all the information gathered during the computation. This information includes the traffic on all communication links and the data received from corrupted and coerced parties. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')$ denote the random variable describing $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}', \vec{r})$ where $\vec{r}$ is uniformly chosen.

Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}', \vec{r})_i$ denote the output of party P_i after running protocol π on input $\vec{x}$, fake input $\vec{x}'$, random input $\vec{r}$, and with a coercive adversary $\mathcal{A}$. Here we assume that the faking algorithm ϕ is specified within π. (By the above convention, we have $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}', \vec{r})_i = \text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}', \vec{r})$ for corrupted parties P_i.) Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')_i$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}', \vec{r})_i$ where $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}') = \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')_1 \ldots \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')_n$.

THE COMPUTATION IN THE IDEAL MODEL. The ideal model represents the 'most we can hope for' from incoercible computation. It is a natural extension of the ideal model used to define adaptive security. In a nutshell, the difference is that here the **ideal-model-adversary** $\mathcal{S}$ can either *corrupt* or *coerce* a party. A corrupted party presents $\mathcal{S}$ with its input while a coerced party presents $\mathcal{S}$ with its fake input. Let us elaborate.

The computation in the ideal model, in the presence of an ideal-model-adversary $\mathcal{S}$, proceeds as follows. The parties have inputs $\vec{x} = x_1, \ldots, x_n$ and fake inputs $\vec{x}' = x'_1, \ldots, x'_n$, and wish to compute $f(x_1, \ldots, x_n)$ where f is a predetermined function. $\mathcal{S}$ has no initial input.

First corruption/coercion stage: First $\mathcal{S}$ proceeds in iterations, where in each iteration $\mathcal{S}$ may decide to corrupt or coerce some party, based on $\mathcal{S}$'s random input and the information gathered so far. If

[3]The fake input is *not* used in the computation in any way, other than being handed to the adversary upon coercion. In fact, one can assume that P_i knows the value of x'_i only upon coercion.

[4]We use the convention that, in the case where $x_i = x'_i$, P_i presents the adversary only with the *used segment* of r_i. (I.e., random bits that were never used are not presented.) Clearly no generality is lost here. This convention is beneficial in that it avoids having the 'fake random input' $r'_i = \phi(x_i, x'_i, r_i, \text{COM}_i)$ contain the random input of ϕ itself: Since in the case of $x_i = x'_i$ algorithm ϕ is not invoked, then the random input of ϕ does not appear in the used segment of r_i. Thus the random input of ϕ should not appear in ϕ's output as well.

[5]The 'devil's advocate' may say that in our model the adversary can always learn the real input of a coerced party by later corrupting it. However, this does not bother us: we do not try to hide the inputs of corrupted parties, only of (coerced) parties which remain uncorrupted. Thus the interesting cases are of adversaries that coerce parties and do *not* later corrupt them.

[6]In fact, our protocols withstand an even stronger adversary model, where even coerced parties follow the instructions of the adversary from the round of coercion on. However, for clarity we stick to the model described above.

a party, P_i, is corrupted then $\mathcal{S}$ learns x_i. If P_i is coerced than $\mathcal{S}$ learns x_i'. Let B denote the set of *corrupted* parties at the end of this stage.

Input substitution stage: $\mathcal{S}$ may alter the inputs of the *corrupted* parties, based on the information gathered so far. Let $\vec{b}$ be the $|B|$-vector of the altered inputs of the corrupted parties, and let $\vec{y}$ be the n-vector constructed from the input $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$.

Computation stage: The parties hand $\vec{y}$ to the trusted party (party P_i hands y_i), and receive $f(\vec{y})$ from the trusted party. In particular, $\mathcal{S}$ now knows the value of $f(\vec{y})$.

Second corruption/coercion stage: Now that the output of the computation is known, $\mathcal{S}$ proceeds in another sequence of iterations, where in each iteration $\mathcal{S}$ may decide to corrupt or coerce some additional party, based on $\mathcal{S}$'s random input and the information gathered so far (this information now includes the value received from the trusted party by parties in B). Also here, if a party P_i is corrupted then $\mathcal{S}$ learns x_i. If P_i is coerced than $\mathcal{S}$ learns x_i'.

Output stage: The uncorrupted parties output $f(\vec{y})$, and the corrupted parties output some arbitrary function of the information gathered by $\mathcal{S}$ during the computation. (This information consists of the inputs of the corrupted parties, the fake inputs of the coerced parties, and the value computed by the trusted party.) We let the n-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{x}') = \langle \text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{x}')_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{x}')_n \rangle$ denote the outputs of *all* the parties on input $\vec{x}$, fake input $\vec{x}'$, a trusted party for computing f, and adversary $\mathcal{S}$ (party P_i outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{x}')_i$).

Say that $\mathcal{S}$ is (c,t)-limited if it coerces up to c parties and corrupts up to t (presumably, but not necessarily, different) parties during each computation.

EQUIVALENCE OF COMPUTATIONS. We are now ready to state the main definition. It requires that executing a secure protocol π for evaluating a function f be equivalent to evaluating f in the ideal model, in the following sense. (We assume that the description of the faking algorithm is part of protocol π.)

Definition 2 *Let f be an n-ary function, π be a protocol for n parties. $\delta : \mathbf{N} \to [0,1]$, we say that π $\delta(n)$-***incoercibly** *(c,t)-***computes** *f if for any (c,t)-limited real-life coercive adversary $\mathcal{A}$, there exists a (c,t)-limited ideal-model-adversary $\mathcal{S}$, such that for every input vector $\vec{x}$ and fake input vector $\vec{x}'$, $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},\vec{x}')$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},\vec{x}')$ are $\delta(n)$-computationally close.*

If $\delta(n)$ is negligible we simply say that π incoercibly (c,t)-computes f

BLACK BOX SIMULATION . In the sequel we use a more restricted notion of equivalence of computations than the one in Definition 2 where the ideal-model adversary is limited to black-box simulation of the real-life adversary. That is, given π there should exist a single ideal-model adversary $\mathcal{S}$ with oracle (or black-box) access to a real-life adversary. This black-box represents the input-output relations of the real-life adversary described above. For concreteness, we present the following description of the "mechanics" of this black-box, representing a real-life adversary. (This description differs from the one in [CFGN] only in that coercion requests are added.) The black-box has a **random tape**, where it expects to find its random input, and an **input-output tape**. Once a special `start` input is given on the input-output tape, the interaction on this tape proceeds in iterations, as follows. Initially, no party is corrupted or coerced. In each iteration l, first the black-box expects to receive all messages sent in the network in the lth round. Next the black-box outputs the messages to be sent by the corrupted parties in the lth round. Next, the black-box may issue several '`corrupt` P_i' and '`coerce` P_i' requests. Both requests should be answered by the input and random input of P_i. Also, a *corrupted* party remains corrupted from this point on. At the end of the interaction, the **view** of the real-life adversary is defined as the contents of the random tape succeeded by the history of the contents of the input-output tape during the entire interaction. Let

$\text{VIEW}_{\mathcal{S},\mathcal{A}}(\vec{x},\vec{x}')$ denote the random variable describing the view of real-life adversary $\mathcal{A}$ in a simulated execution carried out by ideal-model adversary $\mathcal{S}$, and where $\mathcal{S}$ interacts with ideal-model parties having input $\vec{x}$ and fake input $\vec{x}'$.

The simulator is restricted to probabilistic polynomial time (where each invocation of the black-box is counted as one operation). Furthermore, we limit the operation of the simulator as follows. We require that the `start` message is sent only once, and that no party is corrupted in the ideal model unless a request to corrupt this party is issued by the black-box. (The only purpose of the technical restrictions imposed on the operation of the simulator is to facilitate proving composition theorems such as Theorem 5).

2.2 Incoercibility with respect to the outputs

The above definition of incoercible protocols assumes that all parties compute a single, common function of the inputs. A much richer model allows each party to compute a different function of the inputs, and keep the computed value secret. (Still, the functions computed by some parties may be identical, or alternatively have constant output.) We capture this by letting the range of the computed function be an n-fold Cartesian product of some domain, where the ith domain element is interpreted as party P_i's private output.

Here the following additional concern arises. In any distributed protocol, there is a (publicly known) round by which each party knows its output of the computation. Call this round the **output round**. (For instance, the output round may be the last round of the computation.) If a party, P_i, is coerced after the output round, then it may be coerced to also reveal its private *output* as well as its input. We want the party to be able to convincingly claim that his output has any arbitrary value, independently of the claimed input value (namely x_i') and of the information seen by the adversary during the computation.

For this purpose, the real-life and the ideal-model computations are modified as follows. First, in both computations, in addition to its input and fake input each party P_i will now have another input, called the **fake output** and denoted y_i'.

In the real-life computation, if P_i is coerced when its output, y_i, is known, then it will present the adversary with both x_i' *and* y_i'. The claimed random input r_i' is now computed as follows. If $x_i' = x_i$ and $y_i' = y_i$ then $r_i' = r_i$. Otherwise, $r_i' = \phi(x_i, x_i', y_i, y_i', r_i, \text{COM}_i)$. (Note that here ϕ has also y_i, y_i' as inputs.)

The ideal model requires two modifications: First, in the computation phase the trusted party hands each party P_i its private output, y_i (instead of just handing the common output to all parties). Next, whenever P_i is *corrupted* in the second corruption/coercion phase the ideal-model-adversary $\mathcal{S}$ learns x_i and y_i. If P_i is *coerced* then $\mathcal{S}$ learns both x_i' and y_i'.

The rest of the description leading to Definition 2 remains unchanged. We remark that the construction described in the sequel is secure even in this more general scenario.

2.3 Examining Definition 2

We examine three aspects of the definition: the validity of the ideal model, the validity of our abstraction of the real-life computation , and the applicability of the method for demonstrating equivalence of computations.

We first claim that the ideal model indeed captures the 'most we can hope for' from incoercible computation. Indeed, the ideal-model-adversary can perform the inevitable matching of the input and output values claimed by coerced parties against the inputs and outputs the adversary already knows and try to find contradictions. (For instance, assume that the computed function evaluates to 1 only if at most a third of the parties have input 0. If the adversary knows that the function value is 1 but half of the parties claim that their input is 0, then certainly some parties are 'lying'. But the adversary has no idea *who* is 'lying'.) We claim, however, that this type of attack is *all* the adversary can do in the ideal model. (In other words, if a party or a set of parties choose to 'lie in a stupid way' then, even in the ideal model, they will be detected. However, as long as the coerced parties choose their fake inputs and outputs in a way that is undetectable given only the inputs and outputs of the corrupted parties, they will indeed remain undetected.)

Next consider our abstraction of the real-life computation. Here the fake input is arbitrary and unrelated to the real input — representing full freedom in choosing 'what to say to a coercer'. Furthermore, the coerced party is required to follow a specific, publicly known algorithm (namely, ϕ) in generating the 'fake random input'. This makes sure that the fake random input is generated based only on information that is available to the coerced party. This also disallows secret, 'subliminal channel type' faking algorithms. Moreover, in case that the fake input equals the real input the coerced party must present the adversary with its *real* random input. This requirement is crucial: it will force the faking algorithm to generate fake inputs that indeed 'look like' real random inputs.

We remark that an apparently similar abstraction of the real-life computation - namely requiring that a coerced party will present the adversary with *both* the real random input and the output of ϕ - runs into problems that we weren't able to solve. In particular, here the party must present also the random input of ϕ itself, which seems problematic. In other words, it seems that a coerced party can either 'prove' that it has one input or 'prove' that it has another input, but it cannot 'prove' both claims at the same time.

The notion of equivalence of computations is standard [MR, Be, BCG, CFGN, C]. Informally it says: 'whatever can be done by an adversary in a real-life computation could also have been done by an adversary in the ideal model'.

3 An impossibility argument

In this section we show that incoercible computation is impossible in the presence of computationally unbounded adversaries. The same argument demonstrates that known constructions for secure multiparty computation (e.g., [GMW2, BGW, CCD, CFGN]) are coercible (i.e., not incoercible) even if the adversary is limited to probabilistic polynomial time. Moreover, the natural strategy of simply using deniable encryption (see Section 4 below) on top of previously known solutions is susceptible to the same difficulties.

COMPUTATIONALLY UNBOUNDED ADVERSARIES, PRIVATE CHANNELS. A simple argument shows that incoercible computation is *impossible* when the adversary is computationally unbounded, even if the communication channels are private.[7] We first present a rough sketch of the argument. Consider a party P (say, with binary input), and assume that the parties engage in a protocol for computing a function that depends on P's input. Then the transcript of all the messages sent and received by P during the protocol must determine P's input. Upon coercion, P is requested to present its entire transcript. Consequently, in order to claim that his input was different than the value used for the computation, P must 'lie' about at least one of its incoming or outgoing messages during the protocol (i.e., P must claim that the value of this message is different than what was really transmitted). Now, if the sender/recipient of this message is corrupted (or coerced) by the adversary, then the adversary will detect P's lie with high probability. Thus, if the adversary corrupts or coerces t randomly chosen parties, it will 'catch' P with probability $O(\frac{t}{n})$.

Consider for illustration the [BGW] construction for computing a function that whose output depends on P_i's input. Here the first step of P_i is sharing its input among the other players using Shamir's secret sharing scheme. That is, P_i chooses a random polynomial $p_i(\cdot)$ of degree t such that $p_i(0) = x_i$ and sends to player P_j the value $p_i(j)$. If P_i tries to lie about his input then he must choose a fake polynomial q_i as

[7] When the channels are private we augment the model of computation by requiring that each corrupted or coerced party P_i present the adversary not only with its (possibly fake) input and random input, but also with its traffic, i.e., with all the messages sent and received by the party in the past. If P_i is corrupted or if $x_i = x'_i$ then the presented traffic is the real one. If P_i is coerced and $x_i \neq x'_i$ then the presented traffic may be fake (i.e., it is generated by the faking algorithm ϕ). This requirement is unnecessary when the channels are not private, since there the adversary already knows the communication.

the sharing polynomial. Since $p_i \neq q_i$ and both are of degree $t < n$ then they must differ in at least one point, so there exists a j such that $p_i(j) \neq q_i(j)$. Even if the adversary has corrupted only one randomly chosen party P_j then the lie is 'detected' with probability $\frac{1}{n}$.

A more precise statement and proof follows. We restrict ourselves to functions where each party has binary input and output; the result can be easily generalized to any domain. Say that a function $f : \{0,1\}^n \to \{0,1\}^n$ is (i)-sensitive if there exist $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$, and an index $j \neq i$ such that $f(\vec{x}_0)_j \neq f(\vec{x}_1)_j$ (where $\vec{x}_d = x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_n$).

Notice that the only functions that are not i-sensitive for all i are constant functions. However, there exist interesting functions (e.g., 1-out-of-2 Oblivious Transfer) which are not i-sensitive for *some* i. Below we prove a slightly stronger statement than claimed above.

We will also need that f is non-trivial in the sense that x_i cannot be inferred from the output value. (If x_i can be inferred from the output value then, by our definition, the ideal-model-adversary will be able to infer x_i and f will be incoercibly computable in a trivial way.) That is, there should exist other input vectors $\vec{x}' = x'_1, \ldots, x'_n$, such that $x'_i \neq d$ and $f(\vec{x}')_j = f(\vec{x}_d)_j$ for some $d \in \{0,1\}$, and such that the ideal-model-adversary will be unable to decide whether the input vector is $\vec{x}_d$ or $\vec{x}'$. (For instance the function $f(x_1, \ldots, x_n) = x_i$ is i-revealing. The function $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ is i-non-revealing for all i.)

Theorem 3 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be i-sensitive and i-non-revealing for some i. Then there exists no protocol π that $o(\frac{c+t}{n})$-incoercibly (c,t)-computes a (non-trivial) f with $c + t > 1$ in the presence of computationally unbounded adversaries, even if the channels are private. This holds even if the only real-life adversaries considered are those that coerce only P_i and corrupt* one *other party at random.*

Proof: We use the following two facts about the statistical distance $d(\cdot,\cdot)$ between distributions.[8]

1. Let $A(D)$ denote the output distribution of a algorithm A whose input is taken from distribution D. Then for any algorithm A and any distributions D, D' we have $d(D, D') \geq d(A(D), A(D'))$. (Note that A may be randomized; however we will only consider deterministic algorithms.)

2. For distributions D, R, let DR denote the distribution that independently picks an element d from D, an element r from R, and appends them together. Then, $d(D, D') = d(DR, D'R)$.

Let the traffic of a party be the listing of all messages sent and received by this party during the computation. Consider an i-sensitive and i-non-revealing function f. Let $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ and j be as in the above definition of sensitivity. For $\vec{r} = r_0, r_1, \ldots, r_n$, let $T^i_d(\vec{r})$ denote P_i's traffic when the parties' inputs are $\vec{x}_d$ and their random inputs are $\vec{r}$ (r_0 is the adversary's random input). Let T^i_d denote the distribution of $T^i_d(\vec{r})$ where $\vec{r}$ is randomly chosen.

We first show that $d(T^i_0, T^i_1)$ is considerable. Let O^j_d denote P_j's output of protocol π on input $\vec{x}_d$ for the parties. Then O^j_d is a random variable over the choices of $\vec{r}$. In other words, O^j_d is determined via a (deterministic) algorithm EVAL on input $\vec{r}$, where $\vec{x}_d$ is fixed throughout. (EVAL is the algorithm that runs the protocol π with the given adversary, inputs and random inputs.) We take the following slightly different look at this process. Let $\vec{r}^{(i)}_d$ denote the vector $\vec{r}_d$ where $T^i_d(\vec{r})$ is replaced for r_i. It now holds that P_j's output is determined via an analogous deterministic algorithm, EVAL$'$, on input $\vec{r}^{(i)}_d$. (EVAL$'$ is similar to eval, with the exception that P_i's protocol is not run, and instead whenever a message from P_i is expected by another party, it is taken form $T^i_d(\vec{r})$. It can be seen that the message sent by the other parties to P_i will also match $T^i_d(\vec{r})$.) It now follows from facts 1 and 2 above that

$$d(T^i_0, T^i_1) = d(\vec{r}^{(i)}_0, \vec{r}^{(i)}_1) \geq d(O^j_0, O^j_1).$$

[8]The statistical distance between distributions D, D' over domain A is $d(D, D') \stackrel{\text{def}}{=} \sum_{a \in A} |\text{Prob}_D(a) - \text{Prob}_{D'}(a)|$.

Now, assume the protocol is correct with probability $p > 1/2$. Since $f(\vec{x}_0)_j \neq f(\vec{x}_1)_j$, it must hold that $d(T_0^i, T_1^i \geq d(O_0^j, O_1^j) \geq 2p - 1 > 0$.

Next we use the fact that $d(T_0^i, T_1^i)$ is considerable to show that upon coercion the real-life adversary will always learn, with considerable probability, whether $x_i' = x_i$. Since f is i-non-revealing then the ideal-model-adversary is unable to decide, in the ideal model, whether $x_i' = x_i$. Thus it will be unable to 'imitate' the behavior of the real-life adversary and the protocol will not be incoercible by our definition.

Consider a coercion of P_i, and assume that $x_i \neq x_i'$. We distinguish two cases. Assume first that P_i presents the adversary, upon coercion, with the real traffic (i.e., with the real values of the messages sent and received in the past). Then the adversary can use the fact that $d(T_0^i, T_1^i)$ is considerable to decide whether P_i's input equals x_i' or not. (This can be done as follows. Let τ be the traffic presented by P_i. If $\mathrm{Prob}_{T_0^i}(\tau) > \mathrm{Prob}_{T_1^i}(\tau)$ then decide that $x_i = 0$, else $x_i = 1$.)

It remains to consider the case where if $x_i \neq x_i'$ then there exists at least one past round and a party P_k such that the value that P_i claims to have sent or received from P_k in that round is different than the actual value transmitted. But now the adversary can proceed as follows. Corrupt t randomly chosen parties, and compare the values sent/received by these parties from/to P_i with the values claimed by P_i. Now, if $x_i = x_i'$ then P_i presents its real traffic, and no mismatches will be found. If $x_i \neq x_i'$ then mismatch will be found with probability t/n. □

THE COMPUTATIONAL SETTING. When the adversary is computationally bounded the above argument no longer holds: P's input can still be inferred from its traffic, but finding it may take super-polynomial time. This holds even if the channels are all public and the adversary can listen in on all communication. Consequently, P does not have to 'lie' about any of its past messages. Instead P can present a different interpretation (i.e., a different random input) that will make the *same* transcript 'look' like it resulted from a different input.

However, known constructions (e.g., [GMW2, CFGN]) do not take advantage of this leeway, and are thus coercible. That is, these constructions do not allow a party to claim that his input was different than what was really used in the computation without 'lying' about at least one of its outgoing or incoming messages. Thus the above impossibility argument applies to such protocols as well.

In the next two sections we describe how we overcome this problem.

4 Tools

We describe two tools used in our construction: deniable encryption and general adaptively secure computation (in fact, only a limited version of the second tool is needed).

4.1 Deniable encryptions

Deniable encryption schemes, recently introduced in [CDNO], are two-party (public key) encryption protocols with the additional property that later, upon coercion, the sender can convincingly 'lie' about the encrypted value. (That is, the sender can present 'fake random input' that is consistent with the original ciphertext and an encrypted value different than the original one.)

For self-containment, and since the [CDNO] work is not yet widely accessible, we present here a sketch of the definition and constructions relevant to this work. (The construction appears in Appendix A.) We remark that our construction of incoercible protocols uses deniable encryption protocols as a general tool and does not rely on particular implementations.

4.1.1 A definition

We concentrate on encryption functions that are deniable with respect to coercing *the sender*. A deniable encryption scheme contains a key-generation algorithm G, an encryption algorithm E and a decryption algorithm D that constitute a standard semantically secure encryption scheme as defined in [GM], with the exception that there may be a negligible probability of decryption error. Furthermore, the sender should have a 'faking algorithm' ϕ_E with the following property. Let $(e, d) = G(1^l, \rho)$ be the encryption and decryption keys, where l is a security parameter and ρ is G's random input. Given $c = E_e(m_1, r_1)$ (that is, given an encryption of a message m_1 with encryption key e and random coins r_1) and a different message m_2, ϕ_E outputs a fake-random-input $\tilde{r}_2$ that makes c 'look like' an honest encryption of m_2.

We now present a more precise definition. (The definition here is slightly more limited than the one in [CDNO]; this suffices for our needs.) With $c = E_e(m, r)$ we denote the encryption of m using random coins r. With $E(m)$ we denote the random variable describing $E(m, r)$ where the probability is over the choices of r and the random input of G.

Definition 4 *Let l be a security parameter. A triplet (G, E, D) is a $\delta(l)$-sender-deniable encryption scheme if the following requirements hold for $(e, d) = G(1^l, \rho)$:*

Correctness: *The probability* $\mathrm{Prob}_r[D_d(E_e(m)) \neq m]$ *is negligible (as a function of l).*

Security: *For any $m_1, m_2 \in M$ (where M is the domain of messages) we have $E_e(m_1) \stackrel{c}{\approx} E_e(m_2)$.*

Deniability: *There exists an efficient 'faking' algorithm ϕ_E having the following property with respect to any $m_1, m_2 \in M$. Let r_1 be randomly chosen, let $c = E_e(m_1, r_1)$, and let $\tilde{r}_2 = \phi_E(e, m_1, m_2, r_1)$. Then, the random variables*

$$(m_2, \tilde{r}_2, c) \quad \text{and} \quad (m_2, r_2, E_e(m_2, r_2)) \tag{1}$$

are $\delta(l)$-computationally close (see Definition 1), where r_2 is randomly chosen.

The right hand side of (1) describes the adversary's view of an honest encryption of m_2 according to E_e. The left hand side of (1) describes the adversary's view when a party falsely claims that c is an encryption of m_2. The definition requires that the adversary cannot distinguish between the two cases with probability more than $\delta(l)$.

We remark that schemes resilient against coercion of the receiver, or simultaneous coercion of both the sender and the receiver, are defined in a similar way. (For this work we only need schemes resilient against coercing the sender.)

In Appendix A we sketch the [CDNO] encryption scheme. Appropriately choosing the security parameter, this scheme is $\frac{1}{l^k}$-sender deniable, for any constant k.

4.2 Adaptively secure computation

Our construction also makes use of protocols for securely computing any function in the presence of adaptive adversaries (namely, adaptively secure protocols). Such protocols were recently constructed in [CFGN], where a definition is also given. Let us shortly review the [CFGN] definition.

Adaptively secure protocols are defined using a similar framework as the one used here. That is, the real-life computation is compared, in the same way as here, with a computation in an ideal model. The ideal model used to show adaptive security is the same as the one here, with the exception that the parties can only be *corrupted*, and there are no fake inputs nor fake outputs. The real-life adversary is modified accordingly. (Ofcourse, adaptively secure protocols are not required to specify a faking algorithm for coerced parties.)

The [CFGN] construction starts with a protocol π that is adaptively secure in the **private channels setting** (i.e., the setting where the adversary hears only messages sent to corrupted parties). Such protocols (for computing any function) can be constructed via, say, the [BGW] construction. Next, each message sent in π is encrypted using a special type of encryption function, called **non-committing encryption**. (non-committing encryptions are the heart of their construction.)[9]

An important (for us) feature of the [CFGN] construction is that its security can be proven via black-box simulation, similar to the one described in Section 2. That is, for any protocol constructed via [CFGN] there exists a simulator (that is, an ideal-model adversary) $\mathcal{S}$ that carries out, with any real-life adversary, a simulated interaction that is indistinguishable ¿from a real one. However, $\mathcal{S}$ does not know how to deal with *coercive adversaries.* That is, it does not know how to deal with coercion requests made by the black-box, and does not generate such requests in the ideal model.

REMARK. In fact, we do not need the full power of the [CFGN] construction. That is, a much weaker version of adaptive security, briefly sketched below, is sufficient. We remark that the [GMW2] construction, as well as the [BGW, CCD, RB] constructions augmented with standard (semantically secure) encryptions, can be shown to have this weaker version of adaptive security. An immediate consequence of this observation is that our construction, combined with ,say [GMW2], is an alternative construction to [CFGN] for obtaining full-fledged adaptively secure protocols.

We sketch this weaker version of adaptive security. The standard definition of adaptive security requires that, *for each input vector $\vec{x}$,* the outputs of the parties in the ideal model will be indistinguishable ¿from their outputs in the real-life computation *when in both computations the parties have the same input $\vec{x}$.* The weaker version still allows adaptive adversaries, but only requires that the ideal-model and real-life computations be equivalent when the parties' inputs are uniformly chosen, and when the inputs of the parties in the ideal model computation are chosen independently of their inputs in the real-life computation.

5 Protocols

In this section we show how to incoercibly compute any function in the computational setting in the presence of a (c,t)-limited coercive adversary, where $c + t < \lceil \frac{n}{2} \rceil$. In Section 5.1 we present our solution for the case where all parties have one common output. An extension that deals with the case of different, private outputs for different parties is described in Section 5.2. An analysis appears in Section 5.3.

5.1 Computing functions with a common output

A naive attempt at constructing incoercible protocols may be to start with a known secure-but-coercible construction (say, [CFGN]), and encrypt each message using deniable encryption. However, the impossibility argument of Section 3 applies to this solution as well: in order to 'lie' about its input, a coerced party has to 'lie' about the plaintext of at least one ciphertext sent in the past. But if the adversary corrupts or coerces the recipient of this ciphertext then the 'lie' is detected.[10]

[9] Non-committing encryptions have similar flavour to deniable encryption (described in Section 4), in that there exist ciphertexts that can be "opened" as encryptions of, say, both '1' and '0'. However, non-committing encryptions are strictly weaker than deniable ones. One main difference, among others, is that in non-committing encryptions the parties *using* the scheme are, in general, unable to generate ciphertexts that can be opened both ways. Such ciphertexts can only be generated in a simulated execution of the protocol (such as the one carried out by the ideal-model adversary). In deniable encryption each ciphertext generated by parties using the scheme has unique decryption, and at the same time can be "opened" in several ways for a coercer.

[10] This argument holds even if the encryption scheme remains deniable when *both* the sender and the receiver are coerced. This is due to the simple fact that the sender and the receiver cannot 'coordinate their stories'. In other words, unless both the sender and the receiver are 'telling the truth', it is likely that the sender will claim that it sent a bit b and the receiver will claim that it received $\bar{b}$.

We get around this problem by using sender-deniable encryptions *with no specified receiver.* Instead, the ciphertext will be broadcasted and the decryption key will be shared among the players using a threshold sharing scheme. This will allow the players to *jointly* use the information contained in the encrypted messages, but will prevent each individual party (or a small set of parties) from decrypting — thus making their coercion useless for the adversary.

A general description of the construction follows. For simplicity we assume that each player P_i holds a private input $x_i \in \{0,1\}$ and that the players want to compute the value $y = f(x_1, \ldots, x_n)$ where f is a given Boolean function. Let us deal for now with the case where all parties have one common output; in Section 5.2 we generalize our solution to the case of a n-valued function, whose i^{th} component is the private output of player P_i. Our protocol is composed of three phases:

Phase 1 The players together generate n key pairs instances $(e_1, d_1), \ldots, (e_n, d_n)$ of a public-key deniable encryption scheme in such a way that the encryption keys $e_1, \ldots, e_n$ are made public and the corresponding secret keys $d_1, \ldots, d_n$ are shared among the parties. This is done in a secure way, using standard protocols for secure multiparty computation (see Section 4.2.)

Phase 2 Each player P_i contributes his input by broadcasting it encrypted via the deniable encryption with the i^{th} key e_i.

Phase 3 Using their private shares of the secret keys, the players can now jointly compute the function. This phase too is performed using a standard secure multiparty computation protocol (see Section 4.2.)

Following is a more detailed description of the protocol. With x_i we denote the private input of player P_i. With $r_i = \langle r_{i,1}, r_{i,2}, r_{i,3} \rangle$ we denote P_i's random input where $r_{i,k}$ is the randomness used by P_i during Phase k of the protocol.

Phase 1: Setting up the encryption functions. Let (G, E, D) be an undeniable encryption scheme with an appropriate security parameter (see Section 4.) That is if G is run on a random input σ it returns $(e, d) = G(\sigma)$, where e is the public key and d is the secret key of an instance of the undeniable encryption scheme, We consider a modification of the key generation algorithm, $\tilde{G}$ that takes as input a (longer) random string ρ and outputs $\tilde{G}(\rho) = (e, d^{(1)}, \ldots, d^{(n)})$ where the $d^{(i)}$'s are the shares of the secret key d according to a $(\lfloor \frac{n}{2} \rfloor, n)$-threshold secret sharing scheme (e.g. Shamir's [Sh].)

Consider the following n-input, n-output function

$$F_1(\rho_1, \ldots, \rho_n) = (e \circ d^{(1)}, \ldots, e \circ d^{(n)})$$

where $(e, d^{(1)}, \ldots, d^{(n)}) = \tilde{G}(\bigoplus_i \rho_i)$ and $\circ$ denotes concatenation.

Let π_1 be the secure protocol computing the function F_1 (constructed e.g. using [CFGN].) The players will perform π_1 over the secret inputs ρ_i in a such a way that the i^{th} output $e \circ d^{(i)}$ is privately known only to player P_i.

In Phase 1 the players will invoke n times protocol π_1 on the following inputs. Partition the random input $r_{i,1}$ of player P_i into n random strings $r_{i,1} = (\rho_{i,1}, \ldots, \rho_{i,n})$. In the k^{th} invocation of protocol π_1 player P_i will have input $\rho_{i,k}$.

Let $e_1, \ldots, e_n$ be the n public keys generated in such a way and let $d_i^{(j)}$ the P_j's share of the secret key d_i.

Phase 2: Input Contribution. Once e_i is public, each player P_i contributes his own input x_i by broadcasting $z_i = E_{e_i}(x_i, r_{i,2})$.

Phase 3: Output Computation. In this phase the actual computation of the common output $y = f(x_1, \ldots, x_n)$ is performed. To do so we run another secure multiparty computation protocol. The secret input of player P_i consists of the shares $\vec{d^{(i)}} = (d_1^{(i)}, \ldots, d_n^{(i)})$. The values $\vec{z} = (z_1, \ldots, z_n)$ are public inputs, known to everybody. The function to be computed is defined as follows:

$$F_3(\vec{z} \circ \vec{d^{(1)}}, \ldots, \vec{z} \circ \vec{d^{(n)}}) = f(D_{d_1}(z_1), \ldots, D_{d_n}(z_n))$$

Let π_3 be a secure protocol (as in Section 4.2) that computes the function F_3. The players will perform π_3 with the secret inputs $\vec{d^{(i)}}$ and with public input $\vec{z}$. This protocol is augmented with a zero-knowledge proof by each player P_i that the input contributed during this phase is actually the share $\vec{d^{(i)}}$ it received in Phase 1. This an NP-statement, so it can be proven in zero-knowledge (say, using [GMW1]).[11]

The faking algorithm. We specify the actions of player P_i upon coercion. First P_i hands his fake input x'_i to the adversary. If $x_i = x'_i$ then P_i also hands his real random input r_i. If $x_i \neq x'_i$ then P_i gives to the adversary a fake random input r'_i computed using a *faking algorithm* ϕ , that is

$$r'_i = \phi(x_i, x'_i, r_i, \mathrm{COM}_i)$$

where COM_i is the communication of P_i up to the round of coercion.

Algorithm ϕ proceeds as follows. Recall that we split r_i in three components $r_i = \langle r_{i,1}, r_{i,2}, r_{i,3} \rangle$ where $r_{i,k}$ is the randomness used by P_i during Phase k. In particular $r_{i,2}$ is the randomness used to encrypt the input in Phase 2. Thus, if $x_i \neq x'_i$, then P_i invokes the faking algorithm of ϕ_E of the deniable encryption scheme to compute

$$r'_{i,2} = \phi_E(e_i x_i, x'_i, r_{i,2})$$

and returns $r'_i = \langle r_{i,1}, r'_{i,2}, r_{i,3} \rangle$ That is, P_i always answers truthfully about his computations in Phase 1 and 3, and is 'lying' only about the content of the encryption z_i (by 'lying' about the value $r_{i,2}$.) This in particular means that the adversary will receive true shares of the decryption key even from coerced players. This is the main reason why we cannot tolerate a (c, t)-limited adversary with $c + t \geq \frac{n}{2}$.

REMARK: The reader may wonder why we use n different encryption keys, one for each player. The protocol appears to be working fine if we have only one key e, whose secret key d is shared among the players. In Phase 2 each player P_i would broadcast $z_i = E_e(x_i)$ and Phase 3 would compute the function

$$F_3(\vec{z} \circ d^{(1)}, \ldots, \vec{z} \circ d^{(n)}) = f(D_d(z_1), \ldots, D_d(z_n))$$

The problem with this is that seeing $z_i = E_e(x_i)$, a corrupted player P_j could conceivably come up with a ciphertext $z_j = E_e(x_j)$ of an input x_j related to x_i in some form (for instance P_j could broadcast $z_j = z_i$). Thus the protocol would cease to be secure even in the standard sense. This is the malleability problem of encryptions functions [DDN]. In our case the problem is solved by using a different encryption key for each party.

COMPLEXITY OF THE CONSTRUCTION. Let π be the secure protocol for the computation of the function f that results from the construction of [CFGN]. Let π' be the incoercible protocol for the computation of the same function that results from our construction above. We analyze the round and computational complexity of the two protocols. Let R be the number of rounds required by π and C the maximum number of computation steps that each player has to perform in π. With R' and C' denote the same quantities for π'. Then it is easy to verify that

$$R' = R + constant_1 \quad \text{and} \quad C' = C + constant_2$$

[11] In fact, the zero-knowledge proof is not necessary if either of [GMW2, CFGN] is used in phases 1 and 3. This is so since the protocols π_1 and π_3 can be combined, in a natural way, into one protocol where each party P_i has "intermediate output" $\vec{d^{(i)}}$, and subsequently an additional input $\vec{z}$. Here the inputs of π_3 are automatically identical to the outputs of π_1, and no proof is necessary.

We present the protocol using separate π_1 and π_3 for generality and clarity. Also, this way it is easy to see that a single invocation of phase 1 (i.e., a single set of keys) can be used in many different computations.

where $constant_1$ and $constant_2$ do not depend on the particular function f. We remark that a single invocation of protocol π_1 (i.e., a single set of keys) suffices for incoercibly performing many different computational tasks, possibly even where each task has different inputs.

Notice however that we cannot say anything in general about the increase in complexity needed to obtain incoercibility. Indeed let π be an extremely efficient protocol that securely computes a function f, possibly not related to the construction of [CFGN]. Although our techniques can be used to make any protocol, including π, incoercible, we cannot claim the same efficiency results outlined above. See elaboration in Section 6.

5.2 Dealing with secret outputs

Assume that the function f is vector-valued, that is $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where y_i is P_i's output to be kept secret. As defined in Section 2.2, here we require that a coerced player be able to 'lie' about his output component independently of his 'lying' about his input. How can we modify our protocol to achieve that? Noticing that the function F_3 in Phase 3 could be defined to be n-valued too, one would think that running Phase 3 in the multi-value mode would suffice. That is, have each component of the function F_3 be revealed privately to each player. Unfortunately this does not work: in all known constructions for secure computation each output component is shared among the players at the end of the protocol. The shares of the i^{th} component are sent encrypted to P_i who privately reconstruct the output. This means that a player could be coerced into revealing his output component. In fact even if we used deniable encryptions in this phase to exchange messages we would fall back into the same problems outlined in section 3.

We get around this problem as follows. During Phase 2 each player P_i chooses a random bit b_i and broadcasts $w_i = E_{e_i}(b_i)$ in addition to $z_i = E_{e_i}(x_i)$. Let $\vec{w} = (w_1, \ldots, w_n)$. Then during Phase 3 the computed function is

$$F_3'(\vec{z} \circ \vec{w} \circ \vec{d^{(1)}}, \ldots, \vec{z} \circ \vec{w} \circ \vec{d^{(n)}}) = f(D_{d_1}(z_1), \ldots, D_{d_n}(z_n)) \oplus (D_{d_1}(w_1), \ldots, D_{d_n}(w_n))$$

where $\oplus$ is bitwise exclusive or. So basically the private output of P_i at the end of Phase 3 is $y_i \oplus b_i$. This will allow P_i to reconstruct y_i. Also, now P_i can easily fake his output component since the encrypted messages at the end of Phase 3 contain shares of $y_i \oplus b_i$, so he can claim that y_i is either 0 or 1 by faking b_i appropriately.

5.3 Analysis of the main protocol

Theorem 5 *Let $n > 1$. Assume trapdoor permutations exist. If a $\delta(n)$-deniable encryption scheme is used, then for all c, t such that $c + t \leq \lfloor \frac{n}{2} \rfloor$ the protocol in Section 5.1 $(n\delta(n) + \epsilon(n))$-incoercibly $(c + t)$-computes any function f of n variables, where $\epsilon(n)$ is negligible.*

Proof (sketch): The proof is based on three axioms: (1) the security of protocol π_1, (2) the deniability of the encryption scheme (G, E, D), (3) the security of protocol π_3. We only sketch the proof of the basic construction, the construction in Section 5.2 is proven in a similar manner.

THE SIMULATOR $\mathcal{S}$. We construct a simulator $\mathcal{S}$ that will simulate, for any black-box representing a real-life coercive adversary $\mathcal{A}$, a set of players running the protocol. Since π_1 and π_3 are secure protocols they have black-box simulators for an adaptive adversary. Let us call them $\mathcal{S}_1$ and $\mathcal{S}_2$.

The simulator $\mathcal{S}$ will run $\mathcal{S}_1, \mathcal{S}_2$ and the faking algorithms ϕ_E roughly as follows. It will run $\mathcal{S}_1$ to simulate Phase 1. Then it will broadcast encryptions of a fixed value (say 0) to simulate Phase 2. It will use the faking algorithm of E to "open" these encryption in the appropriate way in case of corruption/coercion of a player by the real-life adversary. It will then use $\mathcal{S}_3$ to simulate Phase 3.

We will show that the resulting view will be indistinguishable from the real-life one. The reason is that a distinguisher between the views of $\mathcal{A}$ and $\mathcal{S}$ could be used to contradict one of our three basic hypotheses (security of π_1, deniability of E, security of π_3). Details below.

DETAILS OF SIMULATION. We will describe the simulation in more detail.

Phase 1. For simplicity assume that Phase 1 is composed of n sequential executions of the protocol π_1 for the computation of n copies of the function F_1. Recall that π_1 is secure, thus it has a simulator $\mathcal{S}_1$ for an adaptive adversary.

$\mathcal{S}$ will start running $\mathcal{S}_1$ playing the real-life adversary for $\mathcal{S}_1$. Since $\mathcal{S}_1$ is an ideal-model adversary, whenever it decides to corrupt a player, he must be given its input. Remember that the input of player P_i during the real-life execution of the protocol π_1 is just a random number $r_{i,1}$ which has no relationship with the real input x_i. So $\mathcal{S}$ runs the simulator $\mathcal{S}_1$ using some random values $r_{i,1}$ of his choice as the inputs of the players. The instructions of $\mathcal{S}_1$ to its oracle are passed to $\mathcal{A}$ as part of the simulated execution.

Whenever $\mathcal{A}$ decides to corrupt (resp. coerce) a player P_i, $\mathcal{S}$ will corrupt (resp. coerce) the same player in the ideal model and receive his true (resp. fake) input x_i (resp. x'_i). Also $\mathcal{S}$ will corrupt P_i in the simulated real-life execution of $\mathcal{S}_1$ which means that all this information is also passed along to $\mathcal{A}$.

Whenever $\mathcal{S}_1$ requests the output of the computation, $\mathcal{S}$ will pass it to him ($\mathcal{S}$ can do that since it knows all the inputs).

Phase 2. $\mathcal{S}$ will broadcast, in the name of each uncorrupted player P_i the encryption $z_i = E_{e_i}(0, r_{i,2})$, where $r_{i,2}$ is chosen at random. Also $\mathcal{S}$ looks at what $\mathcal{A}$ broadcasts in the name of the corrupted parties. If P_j is a corrupted party at the beginning of this phase and $\mathcal{A}$ broadcasts z_j, $\mathcal{S}$ will decrypt z_j to obtain the corresponding cleartext x_j and will modify P_j's input accordingly in the ideal model. ($\mathcal{S}$ can decrypt z_j since it knows all the decryption keys.) If $\mathcal{A}$ corrupts (resp. coerces) P_i at the end of Phase 2, $\mathcal{S}$ will receive the real (resp. fake) input x_i (resp. x'_i) by corrupting (resp. coercing) P_i in the ideal model. If the received input is 0 then $\mathcal{S}$ will add $r_{i,2}$ to $\mathcal{A}$'s view. Otherwise $\mathcal{S}$ will invoke the faking algorithm ϕ_E to add $r'_{i,2} = \phi_E(e_i, 0, 1, r_{i,2})$ to $\mathcal{A}$'s view .

Phase 3. Recall that Phase 3 is composed of an execution of the protocol π_3 for the computation of the function F_3, augmented with a zero-knowledge proof in which the players prove that they are using the output of phase 1 as input of π_3. Since π_3 is secure, it has a simulator $\mathcal{S}_3$. In fact, let us denote with $\mathcal{S}_3$ the "augmented" simulator that also run the simulator for this zero-knowledge proof.

$\mathcal{S}$ will run $\mathcal{S}_3$ playing the real-life adversary for $\mathcal{S}_3$. $\mathcal{S}$ runs the simulator $\mathcal{S}_3$ on the output of the simulation of Phase 1 plus $z_1, \ldots, z_n$.

The simulation continues as in Phase 1. Whenever $\mathcal{A}$ decides to corrupt (resp. coerce) a player P_i, $\mathcal{S}$ will corrupt (resp. coerce) the same player in the ideal model and receive his true (resp. fake) input x_i (resp. x'_i). Also $\mathcal{S}$ will corrupt P_i in the simulated real-life execution of $\mathcal{S}_3$ which means that all this information is also passed along to $\mathcal{A}$.

When $\mathcal{S}_3$ requests the output of the computation $\mathcal{S}$ will invoke the trusted party in the ideal model and will forward the response to $\mathcal{S}_3$.

ANALYSIS OF $\mathcal{S}$. Assume that the encryption scheme E is $\delta(n)$-deniable. Under this assumption we will show that $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, \vec{x}')$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')$ are $(n\delta(n) + \epsilon(n))$-computationally close, where $\epsilon(n)$ is a negligible function. To this purpose, it will suffice to show that the *views* of $\mathcal{A}$ in the simulated and real-life execution are $(n\delta(n) + \epsilon(n))$-computationally close (recall definition of view in Section 2.1.)

Assume for contradiction that there exists a polytime distinguisher Dist such that

$$|\text{Prob}(\mathsf{Dist}(\text{VIEW}_{\mathcal{S},\mathcal{A}}(\vec{x}, \vec{x}')) = 1) - \text{Prob}(\mathsf{Dist}(\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')) = 1) \geq n\delta(n) + \epsilon(n)$$

Let R be the number of rounds of the protocol. For $i = 0, \ldots, R$, let H_i be the hybrid probability distribution obtained as follows:

- choose a random view in $\text{VIEW}_{\mathcal{S},\mathcal{A}}(\vec{x}, \vec{x}')$
- choose a random view in $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')$
- take the first i rounds of the view in $\text{VIEW}_{\mathcal{S},\mathcal{A}}(\vec{x}, \vec{x}')$ and the last $R - i$ rounds of the view in $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')$ and paste them together

Notice that $H_0 = \text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')$ and $H_R = \text{VIEW}_{\mathcal{S},\mathcal{A}}(\vec{x}, \vec{x}')$, thus

$$\begin{aligned}&\sum_i |\text{Prob}(\mathsf{Dist}(H_i) = 1) - \text{Prob}(\mathsf{Dist}(H_{i+1}) = 1)| \geq \\ &\qquad |\text{Prob}(\mathsf{Dist}(\text{VIEW}_{f,\mathcal{S}}(\vec{x}, \vec{x}')) = 1) - \text{Prob}(\mathsf{Dist}(\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{x}')) = 1)| \geq n\delta(n) + \epsilon(n) \qquad (2)\end{aligned}$$

Let $1 < k < R$ be the round at the beginning of Phase 2. If $i \neq k$ then it must be that $H_i \stackrel{c}{\approx} H_{i+1}$. This is because for $i \neq k$ the i^{th} round falls into either the secure protocol π_1 or the secure protocol π_3. If it were possible to distinguish between H_i and H_{i+1} then such distinguisher could be used to construct a distinguisher between the view of a real-life adversary $\mathcal{A}$ and the view of the ideal-model adversary $\mathcal{S}_1$ for π_1 (if $i < k$.) For $i > k$ it would be possible to distinguish between $\mathcal{A}$ and $\mathcal{S}_3$). In particular this holds for the distinguisher Dist, i.e. by choosing the ϵ function appropriately

$$|\text{Prob}(\mathsf{Dist}(H_i) = 1) - \text{Prob}(\mathsf{Dist}(H_{i+1}) = 1)| < \frac{\epsilon(n)}{R} \qquad (3)$$

By plugging (3) into (2) we get that for $i = k$

$$|\text{Prob}(\mathsf{Dist}(H_i) = 1) - \text{Prob}(\mathsf{Dist}(H_{i+1}) = 1)| > n\delta(n)$$

But this contradicts the hypothesis that E is a $\delta(n)$-sender-deniable encryption scheme (see Definition 4.) This is because the only difference between H_i and H_{i+1} is that the encrypted messages z's are open in different ways, using either true coin tosses or the faking algorithms ϕ_E. Since there are n such messages and the encryption scheme is $\delta(n)$-deniable, the contradiction follows. □

6 Open problems

In this work we incorporate resilience to coercion in the framework of multiparty computation and demonstrate that any function can be computed in an incoercible way. Still, important problems remain open. Perhaps the most acute one is whether deniable encryption with only negligible probability of successful coercion can be constructed.

Another goal is improving the resilience of our constructions with respect to the number of *coerced* parties. Note that our constructions work even if the coerced parties follow the instructions of the adversary from the point of coercion on. This apparent strength is also a weakness: it means that our approach cannot yield incoercible protocols when more than half of the parties are coerced. Still, it may be the case that incoercible protocols exist even when almost all parties are coerced (and less than half are corrupted).

Finally, we address efficiency. A slight modification of our construction can be presented as a "compiler" that takes any adaptively secure protocol π and turns it into an incoercible protocol π'. For a natural class of protocols (that includes all known general constructions) our construction only results in *additive* increase in the (round, message, and computation) complexity of the original protocol π.[12] But there may exist protocols whose complexity would increase by a large multiplicative factor. It would be nice to have a construction that ensures moderate increase in complexity for *any* protocol.

[12]This is the class of protocols where the input value of each party is shared among the parties in the first round, and only the shares are used in subsequent rounds.

References

[Be] D. Beaver, "Foundations of Secure Interactive Computing", *CRYPTO,* 1991.

[BT] J. Benaloh and D. Tunistra, "Receipt-Free Secret-Ballot Elections", *26th STOC,* 1994, pp. 544-552.

[BCG] M. Ben-Or, R. Canetti and O. Goldreich, "Asynchronous Secure Computations", *25th STOC,* 1993, pp. 52-61.

[BGW] M. Ben-Or, S. Goldwasser and A. Wigderson, " Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th STOC,* 1988, pp. 1-10.

[C] R. Canetti, "Studies in Secure Multiparty Computation and Applications", Ph.D. Thesis, Weizmann Institute of Science, 1995. Available on-line at http://theory.lcs.mit.edu/˜canetti

[CDNO] R. Canetti, C. Dwork, M. Naor and R. Ostrovsky, "Deniable Encryptions", manuscript. Available at the Theory of Cryptography Library, `http://theory.lcs.mit.edu/t̃cryptol/`

[CFGN] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th STOC,* 1996. Also in MIT LCS TR No. 682, 1996.

[CCD] D. Chaum, C. Crepeau and I Damgard, "Multiparty unconditionally secure protocols", *20th STOC,* 1988, pp. 11-19.

[DDN] D. Dolev, C. Dwork, M. Naor, "Non-malleable Encryption", *23th STOC*, 1991, pp. 542-552.

[GL] O. Goldreich and L. Levin, "A Hard-Core Predicate to any One-Way Function", *21st STOC,* 1989, pp. 25-32.

[GMW1] O. Goldreich, S. Micali and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design", *27th FOCS,* 1986, pp. 174–187. Journal version in *JACM*, vol.38, no.1, pp.691-729, 1991.

[GMW2] O. Goldreich, S. Micali and A. Wigderson, "How to Play any Mental Game", *19th STOC,* 1987, pp. 218-229.

[GM] S. Goldwasser and S. Micali, "Probabilistic encryption", *JCSS,* Vol. 28, No 2, April 1984, pp. 270-299.

[He] A. Herzberg, Rump session presentations at CRYPTO'91.

[MR] S. Micali and P. Rogaway, "Secure Computation", in preparation. Preliminary version in *CRYPTO 91.*

[NR] V. Niemi and A. Renvall, "How to prevent buying of votes in computer elections", ASYACRYPT 1994, pp. 141–148.

[RB] T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Secure Computation with Honest Majority", *21th STOC,* 1989, pp. 73–85.

[SK] K. Sako and J. Kilian, "Receipt-Free Mix-Type Voting Scheme", *Eurocrypt* 1995, pp. 393-403.

[Sh] A. Shamir, "How to share a secret", *CACM*, vol.22, Nov. 1979, pp. 612–613.

[Y] A. Yao, "Protocols for Secure Computation", *23th FOCS,* 1982, pp.160–164.

A A deniable encryption scheme

We describe a $\frac{1}{l}$-sender-deniable scheme (based on the existence of trapdoor permutations).[13] This of course means that the probability of successful coercion can be made arbitrarily close to zero by choosing large enough l; still, this rate of decrease leaves much to be desired. An interesting open question is whether there exist schemes that comply with Definition 4 with negligible $\delta(l)$.

We remark that this slow rate of decrease in the probability of coercion propagates to our construction. Any improvement in this rate will automatically translate to a similar improvement in the quality of our construction.

SPARSE SETS WITH TRAPDOOR. The [CDNO] scheme is based on a combinatorial primitive that can be informally described as follows. It is a set $\mathcal{S} \subset \{0,1\}^t$, together with trapdoor information d, such that:
1. $\mathcal{S}$ is small: $|\mathcal{S}| \leq 2^{t-k}$ for some sufficiently large k.
2. It is easy to generate random elements $x \in \mathcal{S}$, even without the trapdoor d.
3. Given $x \in \{0,1\}^t$ and d it is easy to decide whether $x \in \mathcal{S}$.
4. Without d it is infeasible to decide whether a given value $x \in \{0,1\}^t$ was uniformly chosen from $\mathcal{S}$ or from $\{0,1\}^t$.

Call such a construct a **sparse set with trapdoor**. (Formal definitions can be extracted from this description.)

Following are two simple constructions of sparse sets with trapdoor. Both use a trapdoor permutation $f : \{0,1\}^s \rightarrow \{0,1\}^s$, and its hard-core bit function $B : \{0,1\}^s \rightarrow \{0,1\}$ (say, use the Goldreich-Levin bit [GL]).

Construction I: Let $t = sk$. Represent each $x \in \{0,1\}^t$ as a vector $x = x_1...x_k$ where each $x_i \in \{0,1\}^s$. Then let $\mathcal{S} = \{x \in \{0,1\}^{sk} \mid \forall i = 1..k,\ B(f^{-1}(x_i)) = 0\}$. Here $|\mathcal{S}| = 2^{(s-1)k} = 2^{t-k}$.

Construction II: Let $t = s + k$. Represent each $x \in \{0,1\}^t$ as $x = x', b_1...b_k$ where $x' \in \{0,1\}^s$ and each $b_i \in \{0,1\}$. Then let $\mathcal{S} = \{x \in \{0,1\}^{s+k} \mid \forall i = 1..k,\ B(f^{-i}(x')) = b_i\}$. Here $|\mathcal{S}| = 2^s = 2^{t-k}$.

It is easy to verify that both constructions satisfy requirements 1 through 4. Construction II is more efficient in that it uses only $t = s + k$ instead of $t = sk$.

THE SCHEME. The public encryption key is a description of a sparse set $\mathcal{S} \subset \{0,1\}^t$ with trapdoor, where the trapdoor is the secret decryption key. Encryption and decryption proceed as follows.

Let $V \in \{\mathcal{S}, \mathcal{R}\}^l$ denote an l-element vector formed as follows. First choose $\hat{V} \in \{0,1\}^l$. Next, if the ith element of $\hat{V}$ is 1 (resp., 0) then the ith element of V chosen uniformly at random from $\mathcal{S}$ (resp., $\{0,1\}^t$). We call elements drawn uniformly from $\mathcal{S}$ (resp., $\{0,1\}^t$) $\mathcal{S}$-elements (resp., $\mathcal{R}$-elements).

Encryption: *To encrypt 0 (resp., 1), choose a random even (resp., odd) $i \in 0..l$, and send a random $V \in \{\mathcal{S}, \mathcal{R}\}^l$ containing i $\mathcal{S}$-elements.*

Decryption: *Output the parity of the number of elements in V that belong to $\mathcal{S}$.*

Opening an encryption honestly: *Reveal the real random choices used in generating V.*

Opening an encryption dishonestly (algorithm ϕ_E): *Let i be the number chosen by the sender in the encryption process. The sender claims that she has chosen $i-1$ rather than i. (Consequently, the parity of i flips.) For this, she chooses a randomly selected $\mathcal{S}$-element in the ciphertext, and claims that it is an $\mathcal{R}$-element (i.e., a random string in $\{0,1\}^t$).*

Theorem 6 *If trapdoor permutations exist then the above scheme is a $\frac{1}{l}$-sender-deniable encryption scheme.*

Proof (sketch): Security of the scheme against eavesdroppers that see only the ciphertext follows from property 4 of sparse sets. The probability of erroneous decryption is at most $l2^{-k}$ (property 1 of sparse

[13] Using appropriate settings of the security parameter it can be said that the scheme is $\frac{1}{l^k}$-sender-deniable for any constant k.

sets). We show deniability. Assume that l is odd, and let x be an encryption of 1 to generate which the sender chose the value i. Then, i was chosen at random from $1, 3, ...l$. Consequently, the value $i-1$ is uniformly distributed in $0, 2, ..., l-1$. Thus, when the sender claims that she has chosen $i-1$, she demonstrates the correct distribution of i for encrypting 0. Thus, cheating in this direction is undetectable (as long as $\mathcal{S}$-elements cannot be distinguished from $\mathcal{R}$-elements). Assume now that x is an encryption of 1. Thus i is chosen uniformly from $0, 2, ..., l-1$. Now, deniability is possible (and undetectable) as long as $i \neq 0$. However, $i = 0$ only with probability $\frac{1}{l}$. $\square$

This scheme can be modified slightly so as to permit deniability with all but negligible probability. In the modified scheme, to encrypt a bit 0 (resp., 1) the vector V is chosen *uniformly* from all vectors in $\{\mathcal{S}, \mathcal{R}\}^l$ with even (resp., odd) number of $\mathcal{S}$-elements. Also here, the only case where cheating is impossible is when the vector $\hat{V}$ is chosen to be 0^l. But now $\hat{V} = 0^l$ with negligible probability. In this variant, however, the statistical distance between the distribution of the number of $\mathcal{S}$-elements in an honestly opened encryption and in a dishonestly opened encryption is as large as $\frac{1}{\sqrt{l}}$.