Adaptively Secure Multi-party Computation

Ran Canetti[*] Uri Feige[†] Oded Goldreich[‡] Moni Naor[§]

November 8, 1995

Abstract

A fundamental problem in the area of secure multi-party computation is how to deal with *adaptive* adversaries (i.e., adversaries that may choose the corrupted parties during the course of the computation), in a setting where the channels are insecure and secure communication is achieved by cryptographic primitives.

It turns out that the power of an adaptive adversary is greatly affected by the amount of information gathered upon the corruption of a party. This amount of information is, in turn, intimately related to the extent to which uncorrupted parties carry out instructions that cannot be externally verified, such as erasing records of past configurations. It has been shown that if the parties are trusted to erase such records, then adaptively secure computation can be carried out using known primitives. However, this total trust in parties may be unrealistic in many scenarios. An important question, open since 1986, is whether adaptively secure multi-party computation can be carried out in this setting, even if no party in thoroughly trusted.

Our main result is an affirmative resolution of this question, for the case where uncorrupted parties follow their protocols with the exception that all records of past configurations are kept. We first propose a novel property of encryption protocols and show that if an encryption protocol enjoying this property is used, instead of a standard encryption scheme, then known constructions become adaptively secure. Next we construct, based on the standard RSA assumption, an encryption protocol that enjoys this property.

[*]LCS, MIT. Supported by a Rothschild post-doctoral fellowship. canetti@theory.lcs.mit.edu.

[†]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. feige@wisdom.weizmann.ac.il.

[‡]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. Currently visiting LCS, MIT. oded@wisdom.weizmann.ac.il.

[§]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. naor@wisdom.weizmann.ac.il.

1 Introduction

Consider a set of parties who do not trust each other, nor the channels by which they communicate. Still, the parties wish to correctly compute some common function of their local inputs, while keeping their local data as private as possible. This, in a nutshell, is the problem of secure multi-party computation. The parties' distrust in each other and in the network is usually modeled via an adversary that corrupts some of the parties. Once a party is corrupted it follows the instructions of the adversary. In particular, all the information known to this party becomes known to the adversary.

An important parameter, which is the focus of this work, is the way in which the corrupted parties are chosen. In the case of non-adaptive adversaries, the set of corrupted parties is arbitrary, but fixed before the computation starts. (Still, the uncorrupted parties do not know the identities of the corrupted parties.) A more general case is where the adversary chooses to corrupt parties during the course of the computation, based on the information gathered so far. We call such adversaries adaptive.

The difference between adaptive and non-adaptive adversaries may be best demonstrated via an example. Consider the following secret sharing protocol, run in the presence of an adversary that may corrupt $t = O(n)$ out of the n parties: *A dealer D chooses at random a small set S of $m = \sqrt{t}$ parties, and shares its secret among these parties using an m-out-of-m sharing scheme. In addition D publicizes the set S.* Intuitively, this scheme lacks in security since S is public and $|S| \ll t$. Indeed, an adaptive adversary can easily find D's secret, *without corrupting D,* by corrupting the parties in S. However, any non-adaptive adversary that does not corrupt D learns D's secret only if S happens to be identical to the pre-defined set of corrupted parties. This happens only with exponentially small probability. Consequently, this protocol is secure in the presence of non-adaptive adversaries.

Protocols for securely computing any function, in several computation models, have been known for a while: Goldreich, Micali and Wigderson have shown how to securely compute any function in the computational setting [GMW]. (In the computational setting all the communication between the parties is seen by the adversary. All parties, as well as the adversary, are restricted to probabilistic polynomial time). Ben-Or, Goldwasser and Wigderson, and independently Chaum, Crepeau and Damgard, have shown how to securely compute any function in the secure channels setting [BGW, CCD]. (In the secure channels setting the adversary cannot eavesdrop on the communication between uncorrupted parties, and is allowed unlimited computational power.) These protocols can be shown secure in the presence of non-adaptive adversaries. In contrary to folklore beliefs, problems are encountered when attempting to prove *adaptive* security of protocols, *even in the secure channels setting*. Additional problems are encountered in the computational setting. Demonstrating, clarifying, and (partially) solving these problems is the focus of this work.

We first pose the following question: To what extent can uncorrupted parties be trusted to carry out instructions that cannot be externally verified, such as erasing local data, or making random choices? This question is intimately related to the power of an adaptive adversary, in both of the above settings, since the adversary may gather additional information when corrupting parties that have locally deviated from the protocol (say, by not erasing data that is supposed to be erased). If uncorrupted parties are trusted to carry out even unverifiable instructions such as erasing local data then adaptively secure computation can be carried out using known primitives [BH]. However, this trust may be unrealistic in many scenarios. We thus consider parties that, even when uncorrupted, internally deviate slightly from their protocols. We call such parties semi-honest. Several degrees of internal deviation from the protocol are examined with the main focus on parties which follow their protocol with the exception that they keep record of the entire computation. We seek protocols that are secure even if the uncorrupted parties are semi-honest rather than honest.

We present the problems encountered in the secure channels setting, and state the amount of internal deviation from the protocol under which adaptively secure protocols are known to exist. (In particular, under these conditions the [BGW, CCD] protocols can be proven adaptively secure.)

Finally we concentrate on the computational setting, and on semi-honest parties that follow their protocols with the exception that no internal data is ever erased. Is adaptively secure computation possible in this scenario? This question has remained open since the result of [GMW].

We answer this question in the affirmative. The problems encountered, and our solution, are presented via the following transformation. It is a folklore belief that any secure protocol in the secure channels setting can be transformed into a secure protocol in the computational setting, by encrypting each message using a standard (semantically) secure encryption scheme. This belief can indeed be turned into a proof, provided that only *non-adaptive* adversaries are considered. Major difficulties are encountered when trying to prove this belief in the presence of adaptive adversaries. We show how these difficulties are overcome if a novel protocol for transmission of encrypted data is used, instead of standard encryption. We call such encryption protocols non-committing. (Standard encryption schemes are not non-committing.) We also construct a non-committing encryption protocol, based on the existence of a primitive called common domain trapdoor systems. This primitive exists under the RSA assumption.

Non-committing encryption can be roughly described as follows. Traditional encryption schemes have the extra property that the ciphertext may serve as a commitment of the sender to the encrypted data. That is, suppose that after seeing the ciphertext, a third party requests the sender to *reveal* the encrypted data, and show how it was encrypted and decrypted. Using traditional encryption schemes it may be infeasible (or even impossible) for the sender to demonstrate that the encrypted data was any different than what was indeed transmitted. (In fact, many times encryption is explicitly or implicitly used for commitment.) In a non-committing encryption scheme the ciphertext cannot be used to commit the sender (or the receiver) to the transmitted data. That is, a non-committing encryption protocol allows a simulator to generate dummy ciphertexts that look like genuine ones, and can be later "opened" as encryptions of either 1 or 0, at wish. We note that communication over absolutely secure channels is trivially non-committing, since the third party sees no "ciphertext".

Our construction of non-committing data transmission requires all parties to participate in the secure transmission of information between two parties. For benefit of other possible applications, we note that our construction can be carried out in two stages; the first stage, which requires the participation of all parties, does not depend on the data to be delivered (which may even be undetermined at this stage), whereas the second stage consists of a single message transmission from the data-sender to the receiver. Our scheme is resilient as long as at least *one* party remains uncorrupted.

Organization. In Section 2 we discuss the problem of adaptive security and our solution to it in more detail. We keep the presentation informal throughout this section. Precise definitions are given in Appendix B. Our constructions for the non-erasing and honest-looking cases are presented in Sections 3 and 4, respectively.

2 Semi-honesty and adaptive security: Informal presentation

In Subsection 2.1 we discuss the question of what can be expected from an honest party, and present several notions of semi-honest parties. In Subsection 2.2 we describe the problems encountered when trying to prove adaptive security of protocols *in the secure channels setting*, and state existing solutions. In Subsection 2.3 we present the additional problems encountered when trying to prove adaptive security of protocols *in the computational setting*, and sketch our solution.

2.1 Semi-honest parties

The problem of adaptively secure computation is intimately related to the following question: To what extent can uncorrupted parties be trusted to carry out instructions that cannot be externally verified, such as erasing local data, or using randomness as instructed? Honest parties internally deviate from their protocol in many real-life scenarios, such as users that keep record of their passwords, stock-market brokers that keep records of their clients' orders, *operating systems* that "free" old memory instead of erasing or take periodic snapshots of the memory (for error recovery purposes), and computers that use pseudorandom generators as their source of randomness instead of truly random bits. Consider for example a protocol in which party A is instructed to choose a random number r for party B, hand r to B, and then to *erase* r from its own memory. Can B be certain that A no longer knows r? Furthermore, can A now convince a third party (or an adversary that later decides to corrupt A) that he no longer knows r?

To address this issue we introduce the notion of a semi-honest party. Such a party "appears as honest" (i.e., seems to be following its protocol) from the point of view of an outside observer; however, internally it may somewhat deviate from the protocol. For instance, a semi-honest party may fail to erase some internal data, or use randomness not as instructed. (However, semi-honest parties do *not* collaborate.) We wish to have protocols that are secure even when parties are not thoroughly trusted, or in other words when the uncorrupted parties are semi-honest rather than honest. We say that a protocol π' is a semi-honest protocol for a protocol π if a party running π' "appears as" an honest party running π. We want the requirements from π to be satisfied *even if the uncorrupted parties are running any semi-honest protocol for* π. (In the sequel we use the terms 'semi-honest parties' and 'semi-honest protocols' interchangeably.)

The difference between computations in the presence of totally honest parties and computations in the presence of semi-honest parties becomes evident in the presence of adaptive adversaries. Consider a party just corrupted by the adversary, during the course of the computation. If the party is totally honest, then the adversary will see exactly the data specified in the protocol (in particular, any data that was supposed to be erased will not be seen). If the party is semi-honest then the adversary may see a great deal of other data, such as all the past random choices of the party and all the messages the party ever received and sent. Therefore, the adversary may be much more powerful in the presence of semi-honest parties. We elaborate on this crucial point in the sequel.

We distinguish three types of semi-honest behavior. The slightest deviation from the protocol is refraining from erasing data. We call such parties honest-but-non-erasing, or in short non-erasing. Non-erasing behavior is a very simple deviation from the protocol, that is very hard to prevent. Even if the protocol is somehow protected against modifications, it is always possible to add an *external* device that copies all memory locations accessed by the protocol to a "safe" memory. This way a record of the entire execution is kept. Such an external device requires no understanding of the internal structure or of the behavior of the protocol. Furthermore, failure to erase data may occur even without intension of the honest party (see the operating system examples above).

A more severe deviation by a semi-honest party consists of executing some arbitrary protocol other than the specified one, with the restriction that no external test can distinguish between such a behavior and a truly honest behavior. We call parties that deviate in this way honest-looking. Honest-looking parties represent "sophisticated" parties that internally deviate from the protocol in an arbitrary way, but are not willing to take any chance that they will *ever* be uncovered (say, by an unexpected audit). Note that honest-looking parties can do other "harmful" things, on top of not erasing data. For instance, assume that some one-way permutation $f : D \overset{\text{1-1}}{\mapsto} D$ is known to all parties. When instructed to choose a value r at random from D, an honest-looking party can instead choose s at random from D and let $r = f(s)$. Thus, the party cannot be trusted to *not* know $f^{-1}(r)$. (Other, more 'disturbing' deviations from the protocols are possible, we elaborate in the sequel.)

An even more permissive approach allows a semi-honest party to deviate arbitrarily from the protocol, as long as its behavior appears honest to parties *executing the protocol.* We stress that other external tests, not specified in the protocol, may be able to detect such a party as cheating. We call such semi-honest parties weakly-honest. Weakly-honest parties represent the most general internal deviation from the protocol that remains undetected by other parties running the protocol.

The focus of our work is mainly on adaptive security in the presence of non-erasing parties. This coincides with the common interpretation of the problem of adaptive security. To the best of our knowledge, honest-looking and weakly-honest parties were not considered before.

2.2 Adaptive security in the secure channels setting

Although the emphasis of this paper is on the computational setting, we first present the state of knowledge, and sketch the problems involved, in the secure channels setting.

The state-of-the-art with respect to adaptive computation in the secure channels setting can be briefly summarized as follows. Adaptively secure protocols for computing any function exist in the presence of non-erasing parties (e.g., [BGW, CCD]). However, in contrast with popular belief, not every *non-adaptively* secure protocol is also *adaptively* secure in the presence of non-erasing parties. Furthermore, current techniques are insufficient for proving adaptive security of any protocol for computing a non-trivial function in the presence of honest-looking parties.

In order to present the extra difficulty in constructing *adaptively* secure protocols, we roughly sketch the standard definition of secure multi-party computation. Our presentation follows [MR, B, GwL, C], while incorporating the notion of semi-honest parties in the definition. The definition follows the same outline in the secure channels setting and in the computational settings.

Background: How is security defined. First an ideal model for secure multi-party computation is formulated. A computation in this ideal model captures "the highest level of security we can expect from a multi-party computation". Next we require that executing a secure protocol π for evaluating some function f of the parties' inputs in the actual real-life setting is "equivalent" to evaluating f in the ideal model, where the meaning of this "equivalence" is explained below.

A computation in the ideal model proceeds as follows. First an ideal-model-adversary chooses to corrupt a set of parties (either adaptively or non-adaptively), learns their input, and possibly modifies it. Next all parties hand their (possibly modified) inputs to an incorruptible trusted party. The trusted party then computes the expected output (i.e., the function value) and hands it back to all parties. At this stage an adaptive adversary can choose to corrupt more parties. Finally, the uncorrupted parties output the value received from the trusted party whereas the corrupted parties output some arbitrary function of the information gathered during the computation.

In the real-life model there exists no trusted party and the parties must interact with one another using some protocol in order to compute any "non-trivial" function. We say that the execution of a protocol π for evaluating f is "equivalent" to evaluating f in the ideal model, if for any adversary $\mathcal{A}$ in the real-life model, there exists an ideal-model-adversary $\mathcal{S}$ that has the same effect on the computation as $\mathcal{A}$, *even though* $\mathcal{S}$ *operates in the ideal model.* That is, on any input, the outputs of the parties after running π in the real-life model in the presence of $\mathcal{A}$ should be distributed equally to the outputs of parties evaluating f in the ideal model in the presence of $\mathcal{S}$. Furthermore, this condition should hold *for any semi-honest protocol* π' *for* π (according to either of the above notions of semi-honesty).

We require that the complexity of $\mathcal{S}$ be comparable to (i.e., polynomial in) the complexity of $\mathcal{A}$. This requirement can be motivated as follows. Machine $\mathcal{S}$ represents "what could have been learned in the ideal model". Thus, security of a protocol can be interpreted as the following statement: "whatever $\mathcal{A}$ can learn in the real-life model, could have been learned in the ideal model *within comparable complexity*". A much weaker (and arguably unsatisfactory) notion of security emerges if the complexity of $\mathcal{S}$ does not depend on that of $\mathcal{A}$. (This holds even in the non-adaptive case; see Appendix A.1.)

Problems with proving adaptive security. A standard construction of an ideal-model-adversary, $\mathcal{S}$, operates via black-box interaction with the real-life adversary $\mathcal{A}$. (The exact "mechanics" of the black-box representing $\mathcal{A}$ are specified in the sequel.) More specifically, let π' be a semi-honest protocol for π. $\mathcal{S}$ runs the black-box representing $\mathcal{A}$ on a simulated interaction with a set of parties running π'. $\mathcal{S}$ corrupts (in the ideal model) the same parties that $\mathcal{A}$ corrupts in the simulated interaction, and outputs whatever $\mathcal{A}$ outputs. From the point of view of $\mathcal{A}$, the interaction simulated by $\mathcal{S}$ should be distributed identically to an authentic interaction with parties running π'. It is crucial that $\mathcal{S}$ be able to run a successful simulation based only on the information available to it in the ideal model, and in particular *without knowing the inputs of uncorrupted parties.* We restrict our presentation to this methodology of proving security of protocols, where $\mathcal{S}$ is restricted to probabilistic polynomial time. We remark that no other proof method is known in this context. In the sequel we often call the ideal-model-adversary $\mathcal{S}$ a simulator.

Following the above methodology, the simulator that we construct has to generate simulated messages from the uncorrupted parties to the corrupted parties. In the non-adaptive case the set of corrupted parties is fixed and known to the simulator. Thus the simulator can corrupt these parties, in the ideal model, before the simulation starts. In the adaptive case the corrupted parties are chosen by the simulated adversary $\mathcal{A}$ as the computation unfolds. Here the simulator corrupts a party, in the ideal model, only when the simulated adversary decides on corrupting that party. Thus the following extra problem is encountered. Consider a currently uncorrupted party P. Since $\mathcal{S}$ does not know the input of P, it may not know which messages should be sent by P to the corrupted parties. Still, $\mathcal{S}$ has to generate some dummy messages to be sent by the simulated P to corrupted parties. When the simulated adversary $\mathcal{A}$ later corrupts P it expects to see P's internal data. The simulator should now be able to present internal data for P that is consistent with P's newly-learned input and with the messages previously sent by P, according to *the particular semi-honest protocol* π' run by P. It turns out that this can be done for the [BGW] protocols for computing any function in the presence of non-erasing parties. Thus, the [BGW] protocols are adaptively secure *in the presence of non-erasing parties*. We stress, however, that not every protocol which is secure against non-adaptive adversaries is also secure against adaptive adversaries.[1]

In face of honest-looking parties. Even more severe problems are encountered when honest-looking parties are allowed, as demonstrated by the following example. Consider a protocol β that instructs each party, on private input σ, to just publicize a uniformly and independently chosen value r in some domain D and terminate. Let f_0, f_1 be a claw-free pair of permutations over D. Then, on input $\sigma \in \{0,1\}$, an honest-looking party can 'commit' to its input by publicizing $f_\sigma(r)$ instead of publicizing r. Now, if this honest-looking variant of β is shown secure via an efficient black-box simulation as described above, then the constructed simulator can be used to find claws between f_0 and f_1. Similar honest-looking protocols can be constructed for the [BGW, CCD] protocols. Consequently, if claw-free pairs of permutations exist then adaptive security of the [BGW, CCD] protocols, *in the presence of honest-looking parties*, cannot be proven via black-box simulation.

2.3 Adaptive security in the computational setting

In this subsection we sketch the extra difficulty encountered in constructing adaptively secure protocols *in the computational setting*, and outline our solution for non-erasing parties. Consider the following folklore methodology for constructing secure protocols in the computational setting. Start with an adaptively secure protocol π *resilient against non-erasing parties in the secure channels setting*, and construct a protocol $\tilde{\pi}$ by encrypting each message using a standard encryption scheme. We investigate the security of $\tilde{\pi}$ in the computational setting.

Proving that $\tilde{\pi}$ is non-adaptively secure. We first sketch how $\tilde{\pi}$ can be shown *non-adaptively* secure in the computational setting, assuming that π is non-adaptively secure in the secure channels setting. Let $\mathcal{S}$ be the ideal-model-adversary (simulator) associated with π in the secure channels setting. (We assume that $\mathcal{S}$ operates via "black-box simulation" of

[1] See example in the third paragraph of the Introduction.

the real-life adversary $\mathcal{A}$ as described above.) We wish to construct, in the computational setting, a simulator $\tilde{\mathcal{S}}$ for $\tilde{\pi}$. The simulator $\tilde{\mathcal{S}}$ operates just like $\mathcal{S}$, with the following exception. In the computational setting the real-life adversary expects to see the ciphertexts sent between uncorrupted parties. (In the secure channels setting the adversary does not see the communication between uncorrupted parties.) Furthermore, the real-life adversary expects that the messages sent to corrupted parties be encrypted. $\tilde{\mathcal{S}}$ will imitate this situation as follows. First each message sent to a corrupted party will be appropriately encrypted. Next, the simulated uncorrupted parties will exchange dummy ciphertexts. (These dummy ciphertexts can be generated as, say, encryptions of the value '0'.) The validity of simulator $\tilde{\mathcal{S}}$ can be shown to follow, in a straightforward way, from the validity of $\mathcal{S}$ and the security of the encryption scheme in use.

Problems with proving adaptive security. When adaptive adversaries are considered, the construction of a simulator $\tilde{\mathcal{S}}$ in the computational setting encounters the following problem which is a more severe version of the problem encountered in the secure channels setting. Consider an uncorrupted party P. Since $\tilde{\mathcal{S}}$ does not know the input of P, it does not know which messages should be sent by P to other *un*corrupted parties.[2] Still, $\tilde{\mathcal{S}}$ has to generate dummy ciphertexts to be sent by the simulated P to uncorrupted parties. These dummy ciphertexts are seen by the simulated adaptive adversary. When the simulated adversary later corrupts P, it expects to see all of P's internal data, as specified *by the semi-honest protocol* π'. Certainly, this data may include the cleartexts of all the ciphertexts sent and received by P in the past, including the random bits used for encryption and decryption, respectively. Thus, it may be the case that some specific dummy ciphertext c was generated as an encryption of '0', and the simulated P now needs to "convince" the adversary that c is in fact an encryption of '1' (or vice versa). This task is impossible if a standard encryption scheme (i.e., an encryption scheme where no ciphertext can be a legal encryption of both '1' and '0') is used.

We remark that Beaver and Haber [BH] have suggested to solve this problem as follows. Instruct each party to *erase* (say, at the end of each round) all the information involved with encrypting and decrypting of messages. If the parties indeed erase this data, then the adversary will no longer see, upon corrupting a party, how past messages were encrypted and decrypted. Thus the problem of convincing the adversary in the authenticity of past ciphertexts no longer exists. Consequently, such "erasing" protocols can be shown adaptively secure in the computational setting. However, this approach is clearly not valid in the presence of semi-honest parties. In particular, it is not known whether the [BH] protocols (or any other previous protocols) are secure in the presence of non-erasing parties.

Sketch of our solution. We solve this problem by constructing, in the multi-party computational setting, an encryption protocol that serves as an alternative to standard encryption schemes, and enjoys an additional property roughly described as follows. The additional property is that one can efficiently generate dummy ciphertexts that can later be "opened" as encryptions of either '0' or '1', at wish. (Here the word 'ciphertext' is used to denote all the information seen by the adversary during the execution of the protocol.) These dummy ciphertexts are different and yet computationally indistinguishable from the valid encryptions of '0' (or '1') produced in a real communication. We call such encryption protocols non-committing.[3]

Let $\mathcal{E}^{(0)}$ (resp., $\mathcal{E}^{(1)}$) denote the distribution of encryptions of the value 0 (resp., 1) in a public-key encryption scheme. For simplicity, suppose that each of these distributions is generated by applying an efficient deterministic algorithm, denoted $A^{(0)}$ (resp., $A^{(1)}$), to a uniformly selected n-bit string.[4] In a traditional encryption scheme (with no decryption errors) the supports of $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are disjoint (alas $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are computationally indistinguishable). In a non-

[2] There is also the easier problem of generating the messages sent by P to corrupted parties. This was the problem discussed in the previous subsection. However, our hypothesis that $\mathcal{S}$ is a simulator for the secure channel model means that $\mathcal{S}$ is able to generate these cleartext messages. Thus, all that $\tilde{\mathcal{S}}$ needs to do is encrypt the messages it has obtained from $\mathcal{S}$.

[3] This "non-committing property" is reminiscent of the "Chameleon blobs" of [BCC]. Those are *commitment* schemes where the recipient of a commitment c can generate by himself de-commitments of c to both 0 and 1. Here we consider *encryption* schemes where an adversary can generate by himself ciphertexts which can be opened both as encryptions of 1 and as encryptions of 0.

[4] This is an over simplification. Actually, each of these algorithms is also given an n-bit encryption key.

committing encryption scheme, the supports of $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are not disjoint but the probability that an encryption (of either '0' or '1') resides in their intersection, denoted I, is negligible. Thus, decryption errors occur only with negligible probability. However, it is possible to efficiently generate a distribution $\mathcal{E}^{\mathrm{amb}}$ which assumes values in I so that this distribution is computational indistinguishable from both $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$.[5] Furthermore, each "ambiguous ciphertext" $c \in I$ is generated together with two random looking n-bit strings, denoted r_0 and r_1, so that $A^{(0)}(r_0) = A^{(1)}(r_1) = c$. That is, the string r_0 (resp., r_1) may serve as a witness to the claim that c is an encryption of '0' (resp., '1'). See Appendix B.4 for a definition of non-commiting encryption protocols.

Using a non-committing encryption protocol, we resolve the simulation problems which were described above. Firstly, when transforming π into $\tilde{\pi}$, we replace every bit transmission of π by an invocation of the non-committing encryption protocol. This allows us to generate dummy ciphertexts for messages sent between uncorrupted parties so that at a later stage we can substantiate for each such ciphertext both the claim that it is an encryption of '0' and the claim that it is an encryption of '1'. We stress that although dummy ciphertexts appear with negligible probability in a real execution, they are computationally indistinguishable from a uniformly generated encryption of either '0' or '1'. Thus, using a non-committing encryption protocol we construct *adaptively secure protocols* for computing any (recursive) function *in the computational model in the presence of non-erasing parties*. Finally, we construct a non-committing encryption protocol based on the intractability of inverting the RSA, or more generally based on the existence of common-domain trapdoor systems (see Definition C.1). Thus, we get

Theorem 2.1 *If common-domain trapdoor systems exist, then there exist secure protocols for computing any (recursive) function in the computational setting, in the presence of non-erasing parties and adaptive adversaries that corrupt less than a third of the parties.*

We remark that, using standard constructs (e.g., [RB]), our protocols can be modified to withstand adversaries that corrupt less than half of the parties.

Dealing with honest-looking parties. We also sketch a solution for the case of honest-looking parties, assuming, in addition to the above, also the existence of a "trusted dealer" at a pre-computation stage. The dealer hands each party P a truly random string r_P, to be used as random input. Next, the dealer hands the other parties shares of r_P, so that a coalition of all parties other than P can reconstruct r_P. These shares enable us to "force" each party to send messages according to the specification of the protocol. We stress that this result does not hold if an initial (trusted) set-up is not allowed.

3 A solution for non-erasing parties

We show that any function can be securely computed in the computational setting, in the presence of adaptive adversaries and non-erasing parties. In Subsection 3.1 we show how, using a non-committing encryption protocol, a simulatable protocol for computing some function f in the computational setting can be constructed from any simulatable protocol for computing f in the secure channels setting. In Subsection 3.2 we present our construction of non-committing encryption. We use the following result, attributed to [BGW, CCD], as our starting point:[6]

Theorem 3.1 *The* [BGW, CCD] *protocols for computing any function of n inputs are* $(\lceil\frac{n}{3}\rceil - 1)$*-securely computable in a simulatable way,* in the secure channels setting*, in the presence of non-erasing parties and adaptive adversaries.*

[5] Consequently, it must be that $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are computationally indistinguishable. Thus, a non-committing encryption scheme is also a secure encryption scheme in the traditional sense.

[6] A proof of this result can be extracted from [C, Chap. 3], which deals with the more involved asynchronous model.

3.1 Adaptively secure computation given non-committing encryption

The following theorem formalizes the discussion in Section 2.3.

Theorem 3.2 *Let f be an n-ary function, $t < n$ and π be a protocol that t-securely computes f in a simulatable way* in the secure channels setting, *in the presence of non-erasing parties and adaptive adversaries. Suppose that $\varepsilon_{s,r}$ is a t-resilient non-committing encryption protocol, resilient to non-erasing parties and adaptive adversaries, for transmission from P_s to P_r. Let $\tilde{\pi}$ be the protocol constructed from π as follows. For each bit σ transmitted by π from party P_s to party P_r, protocol $\tilde{\pi}$ invokes a copy of a $\varepsilon_{s,r}$ for transmitting σ. Then $\tilde{\pi}$ t-securely computes f, in a simulatable way* in the computational setting, *in the presence of non-erasing parties and adaptive adversaries.*

Proof (sketch): See Appendix D.1. □

3.2 Constructing non-committing encryption

Before describing our non-committing encryption protocol, let us note that one-time-pad is a valid non-committing encryption protocol.[7] The drawback of this trivial solution is that it requires an initial set-up in which each pair of parties share a random string of length at least the number of bits they need to exchange. We remark that the above solution does not seem to extend to the case one uses pseudorandom sequences (instead of random ones) as one-time pads. In any case, an initial set-up in which parties share secret information is not desirable in practice and does not resolve the theoretically important problem of dealing with a setting in which *no secret information is shared a-priori*.

Our scheme consists of two stages. The first stage, in which all parties participate, consists of key generation and distribution. This stage is independent of the bit to be transmitted (and can be performed before this bit is even determined). The second stage, in which the actual transmission takes place, consists of only one message from the sender to the receiver. The scheme uses a collection of trapdoor permutations together with a corresponding hard-core predicate [BM, Y, GrL]. Actually, we need a collection of trapdoor permutation with the additional property that they are many permutations over the same domain. Furthermore, we assume that given a permutation f over a domain D (but not f's trapdoor), one can efficiently generate at random another permutation over D together with a corresponding trapdoor. We stress that the ability to generate a new permutation (together with a corresponding trapdoor) over the same domain should not interfere with the hypothesis that the given permutation is hard to invert (without the trapdoor). Such a collection is called a common-domain trapdoor system, and is formally defined in Appendix C. Popular trapdoor permutations can be formulated in a way which essentially meets the requirements of a common-domain trapdoor system. A specific implementation based on the RSA is given in Appendix C.

In the sequel we refer to common-domain trapdoor systems in a less formal way. We say that two one-way permutations, f_a and f_b, are a pair if they are both permutations over the same domain (i.e., $a = (\alpha, \beta_1)$ and $b = (\alpha, \beta_2)$, where the domain is D_α). We associate the permutations with their descriptions (and the corresponding inverse permutations with their trapdoors). Finally, as stated above, we augment any common-domain trapdoor system with a hard-core predicate, denoted B. (That is, B is polynomial-time computable, but given (f_a and) $f_a(x)$ is it infeasible to predict $B(x)$ with non-negligible advantage over $1/2$.)

Outline of our scheme. In our encryption scheme, the (public) encryption key is a pair (f_a, f_b) of one-way permutations. The (private) decryption key is f_r^{-1} (i.e, the trapdoor of f_r), where r is uniformly distributed in $\{a, b\}$. The simulator will know both f_a^{-1} and f_b^{-1}, and will generate dummy ciphertexts that 'look kosher' as long as only one of f_a^{-1}, f_b^{-1} is known. The simulator will be able to open these dummy ciphertexts as either '0' or '1' by claiming that the description key held by the receiver is either f_a^{-1} or f_b^{-1}. The correspondence between $\{0, 1\}$ and $\{a, b\}$ will be chosen at random

[7] See Appendix A.2.

by the simulator (and never revealed). We first present, in Subsection 3.2.1, the encryption and decryption algorithms as well as observations that will be instrumental for the simulation. In Subsection 3.2.2 we present the key generation protocol. Finally we show that these together constitute the desired non-committing encryption protocol.

3.2.1 Encryption and decryption

Throughout this section we assume that f_a and f_b are randomly selected permutations over the domain D. Recall that B is a hard-core predicate associated with them. The scheme uses a security parameter, k, which can be thought to equal $\log_2 |D|$.

Encryption: to encrypt a bit $\sigma \in \{0, 1\}$ with encryption key (f_a, f_b), the sender proceeds as follows:
First it chooses $x_1, \ldots, x_{8k}$ at random from D, so that $B(x_i) = \sigma$ for $i = 1, ..., 5k$ and $B(x_i) = 1 - \sigma$ otherwise (i.e., for $i = 5k + 1, ..., 8k$). For each x_i it computes $y_i = f_a(x_i)$. These x_i's (and y_i's) are associated with f_a (or with a). Next, it repeats the process with respect to f_b. That is, $x_{8k+1}, \ldots, x_{16k}$ are chosen at random from D, so that $B(x_i) = \sigma$ for $i = 8k + 1, ..., 13k$ and $B(x_i) = 1 - \sigma$ otherwise, and $y_i = f_b(x_i)$ for $i = 8k + 1, ..., 16k$. The latter x_i's (and y_i's) are associated with f_b (or with b). Finally, apply a random re-ordering (i.e., permutation) $\phi : [16k] \to [16k]$ to $y_1, \ldots, y_{16k}$ and send the resulting vector, $y_{\phi(1)}, \ldots, y_{\phi(16k)}$, to the receiver.

Decryption: upon receiving the ciphertext $y_1, \ldots, y_{16k}$, when having private key f_r^{-1} (where $r \in \{a, b\}$), the receiver computes $B(f_r^{-1}(y_1)), \ldots, B(f_r^{-1}(y_{16k}))$, and outputs the majority value among these bits.

Correctness of decryption. Let us first state a simple technical claim.

Claim 3.3 *For all but a negligible fraction of the α's and all but a negligible fraction of permutation pairs f_a and f_b over D_α,*

$$|\text{Prob}(B(f_b^{-1}(f_a(x))) = B(x)) - \frac{1}{2}| \quad \textit{is negligible} \tag{1}$$

where the probability is taken uniformly over the choices of $x \in D_\alpha$.

Proof: See Appendix D.2 □

From this point on, we assume that the pair (f_a, f_b) satisfies Eq. (1).

Lemma 3.4 *Let $\vec{y} = y_1, \ldots, y_{16k}$ be an encryption of a bit σ. Then with probability $1 - 2^{-\Omega(k)}$ the bit decrypted from $\vec{y}$ is σ.*

Proof: See Appendix D.3 □

Simulation assuming knowledge of both trapdoors. In Lemma 3.5 (below) we show how the simulator, knowing the trapdoors of both f_a and f_b, can generate "dummy ciphertexts" $\vec{z} = z_1, \ldots, z_{16k}$ that can be later "opened" as encryptions of both 0 and 1. Essentially, the values $B(f_a^{-1}(z_i))$ and $B(f_b^{-1}(z_i))$ for each z_i are carefully chosen so that this "cheating" is possible. We use the following notations. Fix an encryption key (f_a, f_b). Let the random variable $\lambda_\sigma = (\sigma; \vec{x}, \phi; \vec{y}; r, f_r^{-1})$ describe a legal encryption and decryption process of the bit σ. That is:

- $\vec{x} = x_1, \ldots, x_{16k}$ is a vector of domain elements chosen as specified in the encryption algorithm (i.e., $x_1, \ldots, x_{5k}$ and $x_{8k+1}, \ldots, x_{13k}$ satisfy $B(x_i) = \sigma$, and $x_{5k+1}, \ldots, x_{8k}$ and $x_{13k+1}, \ldots, x_{16k}$ satisfy $B(x_i) = 1 - \sigma$).
- ϕ is a random permutation on $[16k]$.
- $\vec{y} = y_1, \ldots, y_{16k}$ is generated from $\vec{x}$ and ϕ as specified in the encryption algorithm (i.e., $y_{\phi(i)} = f_a(x_i)$ for $1 \le i \le 8k$, and $y_{\phi(i)} = f_b(x_i)$ for $8k < i \le 16k$).
- r is uniformly chosen in $\{a, b\}$ and f_r^{-1} is the inverse of f_r. (Note that the decrypted bit is defined by the majority of the bits $B(f_r^{-1}(y_i))$.)

We remark that the information seen by the adversary, after the sender and receiver are corrupted, includes either λ_0 or λ_1 (but not both).

Lemma 3.5 *Let (f_a, f_b) be the public key, and assume that both f_a^{-1} and f_b^{-1} are known. Then it is possible to efficiently generate $\vec{z}, \vec{x}^{(0)}, \vec{x}^{(1)}, \phi^{(0)}, \phi^{(1)}, r^{(0)}, r^{(1)}$, such that:*

1. $(0; \vec{x}^{(0)}, \phi^{(0)}; \vec{z}; r^{(0)}, f_{r^{(0)}}^{-1}) \overset{c}{\approx} \lambda_0$.
2. $(1; \vec{x}^{(1)}, \phi^{(1)}; \vec{z}; r^{(1)}, f_{r^{(1)}}^{-1}) \overset{c}{\approx} \lambda_1$.

Note that the same dummy ciphertext, $\vec{z}$, appears in both (1) and (2).

Proof: See Appendix D.4 □

3.2.2 Key generation

In Figure 1 we describe our key generation protocol. This protocol is valid as long as at least one party remains uncorrupted. Our protocol uses a particular implementation of Oblivious Transfer described in [GMW] (which in turn originates in [EGL]). For self containment we sketch this implementation in Appendix E. This implementation has an additional property, discussed in Appendix E, that is useful in our construction. Loosely speaking, Oblivious Transfer (OT) [R, EGL] is a two-party protocol for a sender S having inputs s_1 and s_2, and a receiver R with input $\tau \in \{1, 2\}$. After executing OT, the receiver should know s_τ, and learn nothing else. The sender S should learn nothing from participating in the protocol. In particular S should not know whether R got s_1 or s_2. We are only concerned with the case where R is uncorrupted and non-erasing.

key-generation (ε_{G})

For generating an encryption key (f_a, f_b) known to the sender, and a decryption key f_r^{-1} known only to the receiver (R), where r is uniformly distributed in $\{a, b\}$.

1. The receiver generates a common domain D_α and sends α to all parties.
2. Each party P_i generates two trapdoor permutations over D_α, denoted f_{a_i} and f_{b_i}, and sends (f_{a_i}, f_{b_i}) to R. The trapdoors of f_{a_i} and f_{b_i} are kept secret by P_i.
3. The receiver R chooses uniformly $\tau \in \{1, 2\}$ and invokes the OT protocol with each party P_i for a number of times equal to the length of the description of the trapdoor of a permutation over α. In all invocations the receiver uses input τ. In the j^{th} invocation of OT, party P_i acting as sender uses input (σ_1, σ_2), where σ_1 (resp., σ_2) is the j^{th} bit of the trapdoor of f_{a_i} (resp., f_{b_i}). (Here we use the convention by which, without loss of generality, the trapdoor may contain all random choices made by G_2 when generating the permutation. This allows R to verify the validity of the data received from P_i.)
4. Let H be the set of parties with which all the OT's were completed successfully. Let f_a be the composition of the permutations f_{a_i}'s for $P_i \in H$, in some canonical order, and let f_b be defined analogously (e.g., a is the concatenation of the a_i with $i \in H$). Let $r = a$ if $\tau = 1$ and $r = b$ otherwise. The trapdoor to f_r is known only to R (it is the concatenation of the trapdoors obtained in Step 3).
5. R now sends the public key (f_a, f_b) to the sender.

Figure 1: The key generation protocol

3.2.3 Simulation (Adaptive security of the encryption protocol)

Let ε denote the combined encryption and decryption protocols, preceded by the key generation protocol.

Theorem 3.6 *Protocol ε is an $(n-1)$-resilient non-committing encryption protocol for n parties, in the presence of non-erasing parties.*

Proof (sketch): See Appendix D.5. □

Theorem 2.1 follows from Theorems 3.1, 3.2 and 3.6.

4 Honest-looking parties

Our construction for honest-looking parties assumes the existence of a "trusted dealer" at a pre-computation stage. The dealer chooses, for each party P, a truly random string r_P, and hands r_P to P, to be used as random input. (We call r_P a certified random input for P.) Next, the dealer generates $n-1$ shares of r_P, so that r_P can be reconstructed from all $n-1$ shares, but any subset of $n-2$ shares are independent of r_P. Finally the dealer hands one share to each party *other than P*.

Now, all parties are able to jointly reconstruct r_P, and thus verify whether P follows its protocol. Consequently, if party P is honest-looking (i.e., P does not take any chance of being caught cheating), then it is forced to use r_P exactly as instructed in the protocol. Party P is now limited to non-erasing behavior, and the construction of Section 3 applies. (We note that the use of certified random inputs does not limit the simulator. That is, upon corruption of party P, the simulator can still compute some convenient value r'_P to be used as P's random input, and then "convince" the adversary that the certified random input of P was r'_P. The adversary will not notice anything wrong since it will never have all the shares of the certified random input.)

References

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

[BH] D. Beaver and S. Haber, "Cryptographic Protocols Provably secure Against Dynamic Adversaries", *Eurocrypt,* 1992.

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

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

[BM] M. Blum, and S. Micali, "How to generate Cryptographically strong sequences of pseudo-random bits", em SIAM J. on Computing, Vol. 13, 1984, pp. 850-864.

[BCC] G. Brassard, D. Chaum and C. Crepeau, "Minimum Disclosure Proofs of Knowledge", *Journal of Computing and System Sciences,* Vol. 37, No. 2, 1988, pp. 156-189.

[C] R. Canetti, "Studies in Secure Multi-Party Computation and Applications", Ph.D. Thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, June 1995.

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

[CK] B. Chor and E. Kushilevitz, "A Zero-One Law for Boolean Privacy", *SIAM J. on Disc. Math., Vol. 4, no. 1,* pp. 36-47, 1991.

[EGL] S. Even, O. Goldreich and A. Lempel, "A randomized protocol for signing contracts", *CACM,* vol. 28, No. 6, 1985, pp. 637-647.

[GGL] O. Goldreich, S. Goldwasser, and N. Linial, "Fault-Tolerant Computation in the Full Information Model", *32nd FOCS,* pp. 447-457, 1991.

[GILVZ] O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan and D. Zuckerman, "Security Preserving Amplification of Hardness", FOCS 1990, pp. 318–326.

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

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

[GwL] S. Goldwasser and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO,* 1990.

[GMR] S. Goldwasser , S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Journal on Comput.,* Vol. 18, No. 1, 1989, pp. 186-208.

[MR] S. Micali and P. Rogaway, "Secure Computation", *CRYPTO,* 1991.

[R] M. Rabin, "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[RB] T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multi-party Protocols with Honest Majority", *21st STOC,* 1989, pp. 73-85.

[RSA] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *CACM*, Vol. 21, Feb. 1978, pp. 120–126.

[Y] A. Yao, "Theory and applications of trapdoor functions", *23rd FOCS,* 1982, pp. 80-91.

A Various remarks

A.1 The importance of comparable complexity

We illustrate the importance of requiring that the simulator's complexity is related to the adversary's complexity via the following example. Let $f(x, y) = g(x \oplus y)$ where g is a one-way permutation and $\oplus$ denotes bitwise exclusive or. Assume that parties A and B have inputs x and y respectively, and consider the following protocol for computing f: Party A announces x, party B announces y, and both parties compute $f(x, y)$. Our intuition is that this protocol is insecure against adversaries that may corrupt one party (say B): it "gives away for free" both x and y, whereas computing x given y and $f(x, y)$, may take the adversary a large amount of time. Indeed, if the ideal-model adversary $\mathcal{S}$ is limited to probabilistic polynomial time (and one way permutations exist), then this protocol is insecure against adversaries that corrupt one party. However, under the model allowing $\mathcal{S}$ unlimited computational power regardless of $\mathcal{A}$'s complexity, this protocol is considered secure since $\mathcal{S}$ can invert g.

A.2 One-time pads are non-committing

Assume that each pair of parties share a sufficiently long secret random string, and each message is encrypted by bitwise xor-ing it with a new segment of the shared random string. Then Definition B.10 is satisfied in a straightforward way. Specifically, the simulated message from the sender to the receiver (i.e., the dummy ciphertext), denoted c, can be uniformly chosen in $\{0, 1\}$. When either the sender or the receiver are corrupted, and the simulator has to demonstrate that c is an encryption of a bit σ, the simulator claims that the corresponding shared random bit was $r = c \oplus \sigma$. Clearly r is uniformly distributed, regardless of the value of σ.

B Definitions

In Section B.1 we define semi-honest protocols (with respect to the three variants discussed in Section 2.1). This notion underlies all our subsequent definitions. In Sections B.2 and B.3 we define adaptively secure multi-party computation in the secure channels and the computational settings, respectively. Although the focus of this work is the computational setting, we state this definition also in the secure channels setting. This will enable us to discuss our results as a general transformation from adaptively secure protocols in the secure channels setting into adaptively secure protocols in the computational setting, without getting into details of specific protocols. In Section B.4 we define our main tool, non-committing encryption protocols. Throughout Appendix B we assume that the reader has acquired the intuition provided in Section 2.

Let us first recall the standard definition of computational indistinguishability of distributions.

Definition B.1 *Let $\mathcal{A} = \{A_x\}_{x\in\{0,1\}^*}$ and $\mathcal{B} = \{B_x\}_{x\in\{0,1\}^*}$ be two ensembles of probability distributions. We say that $\mathcal{A}$ and $\mathcal{B}$ are* computationally indistinguishable *if for every positive polynomial p, for every probabilistic polynomial-time algorithm D and for all sufficiently long x's,*

$$|\text{Prob}(D(A_x) = 1) - \text{Prob}(D(B_x) = 1)| < \frac{1}{p(|x|)}.$$

We colloquially say that "A_x and B_x are computationally indistinguishable", or "$A_x \overset{c}{\approx} B_x$".

B.1 Semi-honest protocols

We define semi-honest parties (or, equivalently, semi-honest protocols) for the three alternative notions of semi-honesty discussed in Section 2.1. First we define honest-but-non-erasing (or in short non-erasing) protocols. Informally, a protocol π' is non-erasing for a protocol π, if π' is identical to π with the exception that π' may omit instructions to erase data. Actually, it suffices to consider a non-erasing protocol which keeps a record of the entire history of the computation.

Definition B.2 *Let π and π' be n-party protocols. We say that π' is a* non-erasing *protocol for π if π' is identical to π with the exception that, in addition to the instructions of π, protocol π' copies the contents of each memory location accessed by π to a special record tape (inaccessible by π).*

Next we define honest-looking protocols. Informally, a party is honest-looking if its behavior is indistinguishable from the behavior of an honest party by any external test. (Internally the party may arbitrarily deviate from the protocol.) More formally, let $\text{COM}_\pi(\vec{x}, \vec{r})$ denote the communication among n parties running π on input $\vec{x}$ and random input $\vec{r}$ (x_i and r_i for party P_i). Let $\text{COM}_\pi(\vec{x})$ denote the random variable describing $\text{COM}_\pi(\vec{x}, \vec{r})$ when $\vec{r}$ is uniformly chosen. For n-party protocols ρ and π and an index $i \in [n]$, let $\rho_{/(i,\pi)}$ denote the protocol where party P_i executes π and all the other parties execute ρ.

Definition B.3 *Let π and π' be n-party protocols. We say that π' is a* perfectly honest-looking *protocol for π if for any input $\vec{x}$, for any n-party "test" protocol ρ, and for any index $i \in [n]$, we have*

$$\text{COM}_{\rho_{/(i,\pi)}}(\vec{x}) \stackrel{\text{d}}{=} \text{COM}_{\rho_{/(i,\pi')}}(\vec{x})$$

(where $\stackrel{\text{d}}{=}$ stands for "identically distributed"). If the test protocol ρ is restricted to probabilistic polynomial time, and $\text{COM}_{\rho_{/(i,\pi)}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{COM}_{\rho_{/(i,\pi')}}(\vec{x})$, then we say that π' is a computationally honest-looking *protocol for π.*

Here the "test" protocol ρ represents a collaboration of all parties in order to test whether P_i is honest.

Next we define weakly-honest protocols. Here we require that Definition B.3 is satisfied only with respect to the original protocol π, rather than with respect to any test protocol ρ.

Definition B.4 *Let π and π' be n-party protocols. We say that π' is a* perfectly weakly-honest *protocol for π if for any input $\vec{x}$ and for any index $i \in [n]$, we have*

$$\text{COM}_\pi(\vec{x}) \stackrel{\text{d}}{=} \text{COM}_{\pi/(i,\pi')}(\vec{x})$$

If π is restricted to probabilistic polynomial time, and if $\text{COM}_\pi(\vec{x}) \stackrel{\text{c}}{\approx} \text{COM}_{\pi/(i,\pi')}(\vec{x})$, then we say that π' is a computationally weakly-honest *protocol for π.*

B.2 Adaptively secure computation in the secure channels setting

We define adaptively secure multi-party computation in the secure channels setting. That is, we consider a synchronous network where every two parties are connected via a secure communication link (i.e., the adversary does not see, nor alter, messages sent between uncorrupted parties). The adversary is computationally unlimited.

We use the standard methodology presented in Section 2.2. That is, the execution of a protocol for computing some function is compared to evaluating the function in an ideal model, where a trusted party is used. We substantiate the definition in three steps. First, we give an exact definition of this ideal model. Next, we formulate our (high level) notion of 'real-life' protocol execution. Finally, we describe and formalize the method of comparing computations.

The computation in the ideal model, in the presence of an ideal-model-adversary $\mathcal{S}$, proceeds as follows. The parties have inputs $\vec{x} = x_1 \ldots x_n \in D^n$ (party P_i has input x_i) and wish to compute $f(x_1, \ldots, x_n)$, where f is a predetermined function.[8] The adversary $\mathcal{S}$ has no initial input, and is parameterized by t, the maximum number of parties it may corrupt.

First corruption stage: First, $\mathcal{S}$ proceeds in up to t iterations. In each iteration $\mathcal{S}$ may decide to corrupt some party, based on $\mathcal{S}$'s random input and the information gathered so far. Once a party is corrupted its internal data (that is, its input) becomes known to $\mathcal{S}$. A corrupted party remains corrupted for the rest of the computation. 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; however, this is done without any knowledge of the inputs of the good parties. 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.[9]

Second corruption stage: Now that the output of the computation is known, $\mathcal{S}$ proceeds in another sequence of up to $t - |B|$ iterations, where in each iteration $\mathcal{S}$ may decide to corrupt 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). We stress that $\mathcal{S}$ may corrupt at most t parties in the entire computation.

Output stage: The uncorrupted parties output $f(\vec{y})$, and the corrupted parties output some arbitrary function, computed by the adversary, of the information gathered by the adversary (i.e., $\vec{b}$ and $f(\vec{y})$). We let the n-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ denote the outputs of the parties on input $\vec{x}$, trusted party for computing f, and adversary $\mathcal{S}$ (party P_i outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i$).

For the benefit of extremely formalistic readers we further formalize the above discussion (in Definitions B.5 through B.7). Other readers are advised to skip a page up to the paragraph discussing the computation in the real-life setting.

First, we need two technical notations.

- For a vector $\vec{x} = x_1 \ldots x_n$ and a set $B \subseteq [n]$, let $\vec{x}_B$ denote the vector $\vec{x}$, projected on the indices in B.
- For an n-vector $\vec{x} = x_1 \ldots x_n$, a set $B \subseteq [n]$, and a $|B|$-vector $\vec{b} = b_1 \ldots b_{|B|}$, let $\vec{x}/_{(B,\vec{b})}$ denote the vector constructed from vector $\vec{x}$ by substituting the entries whose indices are in B by the corresponding entries from $\vec{b}$.

Definition B.5 *Let D be the domain of possible inputs of the parties, and let $\mathcal{R}$ be the domain of possible random inputs. A t-*limited ideal-model-adversary *is a quadruple $\mathcal{S} = (t, b, h, O)$, where:*

- *t is the maximum number of corrupted parties.*
- *$b : [n]^* \times D^* \times \mathcal{R} \to [n] \cup \{\perp\}$ is the* selection *function for corrupting parties (the value $\perp$ is interpreted as "no more parties to corrupt at this stage")*
- *$h : [n]^* \times D^* \times \mathcal{R} \to D^*$ is the* input substitution *function*
- *$O : D^* \times \mathcal{R} \to \{0, 1\}^*$ is an* output *function for the bad parties.*

[8] A more general formulation allows different parties to compute a different functions of the input. Specifically, in this case the range of f is a n-fold Cartesian product and the interpretation is that the i^{th} party should get the i^{th} component of $f(\vec{x})$.

[9] In the case where each party computes a different function of the inputs, as discussed in the previous footnote, the trusted party will hand each party its specified output.

The set of corrupted parties is now defined as follows.

Definition B.6 *Let D be the domain of possible inputs of the parties, and let $\mathcal{S} = (t, b, h, O)$ be a t-limited ideal-model-adversary. Let $\vec{x} \in D^n$ be an input vector, and let $r \in \mathcal{R}$ be a random input for $\mathcal{S}$. The i*th set of faulty parties *in the ideal model $B^{(i)}(\vec{x}, r)$, is defined as follows.*

- $B^{(0)}(\vec{x}, r) = \phi$
- *Let $b_i \triangleq b(B^{(i)}(\vec{x}, r), \vec{x}_{B^{(i)}(\vec{x},r)}, r)$. For $0 \leq i < t$, and as long as $b_i \neq \perp$, let*

$$B^{(i+1)}(\vec{x}, r) \triangleq B^{(i)}(\vec{x}, r) \cup \{b_i\}$$

- *Let i^* be the minimum between t and the first i such that $b_i = \perp$. Let $b_i^f \triangleq b(B^{(i)}(\vec{x}, r), \vec{x}_{B^{(i)}(\vec{x},r)}, f(\vec{y}), r)$, where $\vec{y}$ is the substituted input vector for the trusted party. That is, $\vec{y} \triangleq \vec{x}/_{(B^{(i^*)}(\vec{x},r), h(B^{(i^*)}(\vec{x},r), \vec{x}_{B^{(i^*)}(\vec{x},r)}, r))}$.*
  For $i^* \leq i < t$, let
  $$B^{(i+1)}(\vec{x}, r) \triangleq B^{(i)}(\vec{x}, r) \cup b_i^f.$$

In Definition B.7 we use $B^{(i)}$ instead of $B^{(i)}(\vec{x}, r)$.

Definition B.7 *Let $f : D^n \to D'$ for some sets D, D' be the computed function, and let $\vec{x} \in D^n$ be an input vector. The* output of computing function f in the ideal model *with adversary $\mathcal{S} = (t, b, h, O)$, on input $\vec{x}$ and random input r, is an n-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ of random variables, satisfying for every $1 \leq i \leq n$:*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i = \begin{cases} f(\vec{y}) & \text{if} \quad i \notin B^{(t)} \\ O(\vec{x}_{B^{(t)}}, f(\vec{y}), r) & \text{if} \quad i \in B^{(t)} \end{cases}$$

where $B^{(t)}$ is the t^{th} set of faulty parties, r is the random input of $\mathcal{S}$, and $\vec{y} = \vec{x}/_{(B^{(t)}, h(B^{(t)}, \vec{x}_{B^{(t)}}, r))}$ is the substituted input vector for the trusted party.

Computation in the real-life setting. Next we describe the execution of a protocol π in the real-life scenario. The parties engage in a synchronous computation in the secure channels setting, running a semi-honest protocol π' for π (according to any one of the notions of semi-honesty defined above). A computationally unbounded t-limited real-life adversary may choose to corrupt parties at any point during the computation, based on the information known to the previously corrupted parties, and as long as at most t parties are corrupted altogether. Once a party is corrupted the current contents of its memory (as determined by the semi-honest protocol π') becomes available to the adversary. From this point on, the corrupted party follows the instructions of the adversary. Once the computation is completed, each uncorrupted party outputs whatever it has computed to be the function value. Without loss of generality, we use the convention by which the corrupted parties output their entire view on the computation. The view consists of all the information gathered by the adversary during the computation. Specifically, the view includes the inputs and random inputs of the corrupted parties and all the communication seen by the corrupted parties.

We use the following notation. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})$ denote the view of the adversary $\mathcal{A}$ when interacting with parties running protocol π on input $\vec{x}$ and random input $\vec{r}$ (x_i and r_i for party P_i), as described above. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})_i$ denote the output of party P_i after running protocol π on input $\vec{x} = x_1 \ldots x_n$ and random input $\vec{r} = r_1 \ldots r_n$, and with a real life adversary $\mathcal{A}$. (By the above convention, we have $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})_i = \text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})$ for corrupted parties P_i.) Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_i$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})_i$ where $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}) = \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_1 \ldots \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_n$.

Comparing computations. Finally we require that executing a secure protocol π for evaluating a function f be equivalent to evaluating f in the ideal model, in the following sense.

Definition B.8 *Let f be an n-ary function, π be a protocol for n parties and $\mathcal{T}$ a type of semi-honest behavior (i.e., as in any of the Definitions B.2 through B.4). We say that π t-*securely computes *f* in the secure channels setting*, in the presence of $\mathcal{T}$-semi-honest parties and adaptive adversaries, if for any $\mathcal{T}$-semi-honest protocol π' for π and for any t-limited real-life (adaptive) adversary $\mathcal{A}$, there exists a t-limited ideal-model-adversary $\mathcal{S}$, such that the complexity of $\mathcal{S}$ is polynomial in the complexity of $\mathcal{A}$, and for every input vector $\vec{x}$ we have*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi',\mathcal{A}}(\vec{x})$$

Remark: Definition B.8 is stated for a single value of n. In order to discuss asymptotic complexity (in n), we assume that the function f, the protocol π, the simulator $\mathcal{S}$ and the adversary $\mathcal{A}$ are Turing machines that have n, the number of parties, as part of their inputs.

Black-box simulation. In the sequel we use a more restricted notion of equivalence of computations, where the ideal-model adversary is limited to black-box simulation of the real-life setting. That is, for any semi-honest protocol π' for π there should exist a 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. The black-box has a random tape, where the black-box 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. In each iteration l, first the black-box expects to receive the information gathered in the lth round. (In the secure channels setting this information consists of the messages sent by the uncorrupted parties to the corrupted parties.) Next 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' requests. Such a request should be answered by the internal data of P_i, according to protocol π'. Also, from this point on P_i is corrupted. At the end of the interaction, the output 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. We let $\mathcal{S}^{\mathcal{A}}$ denote the ideal-model adversary $\mathcal{S}$ with black-box access to a real-life adversary $\mathcal{A}$.

The simulator is restricted to probabilistic polynomial time (where each invocation of the black-box is counted as one operation).[10] 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.

If Definition B.8 is satisfied by an ideal-model adversary limited to black-box simulation as described above, then we say that π t-securely computes f in a simulatable way. In this case we call the ideal-model adversary a black-box simulator, or in short a simulator.

We remark that the only purpose of the technical restrictions imposed on the operation of the simulator is to facilitate proving composition theorems (such as Theorem 3.2). We stress that the security of known protocols (e.g., [BGW]) can be shown via simulators that obey these restrictions.

B.3 Adaptively secure computation in the computational setting

We now turn to define adaptively secure multi-party computation in the computational setting. Here the communication links between parties are insecure; that is, all messages sent on all links are seen by the adversary. [11] All parties, as

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

[11] For simplicity we assume that the links are authenticated, namely the adversary cannot *alter* the communication. Authenticity can be achieved via standard primitives.

well as the adversary, are restricted to probabilistic polynomial time. Furthermore, we introduce a security parameter, determining 'how close' a real-life computation is to a computation in the ideal model. All parties are polynomial also in the security parameter. For simplicity of presentation, we identify the security parameter and the length of the inputs with the number of parties, denoted n.

The framework of defining adaptively secure multi-party computation in this setting is the same as in the secure channels setting (Section B.2). That is, we compare the real life computation with a computation in the same ideal model. Since the real-life adversary is restricted to probabilistic polynomial time, so is the ideal-model adversary. The execution of a protocol π in the real-life scenario (of the computational setting), as well as the notation $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$, are the same as in the secure channels setting, with the exception that the real-life adversary sees all the communication between the uncorrupted parties. Needless to say that the ideal model is the same in both settings.

We define equivalence of a real-life computation to an ideal-model computation in the same way, with the exception that here we only require that the corresponding distributions are computationally indistinguishable. Black-box simulation is defined as in the secure channels setting, with the exception that the information gathered by the adversary in each round includes the communication between all parties.

Definition B.9 *Let f be an n-ary function, π be a protocol for n parties and $\mathcal{T}$ a type of semi-honest behavior (i.e., as in any of the Definitions B.2 through B.4). We say that π t-*securely computes f in the computational setting*, in the presence of $\mathcal{T}$-semi-honest parties and adaptive adversaries, if for any $\mathcal{T}$-semi-honest protocol π' for π and for any t-limited real-life (adaptive) adversary $\mathcal{A}$, there exists a t-limited ideal-model-adversary $\mathcal{S}$, such that for every input vector $\vec{x}$ we have*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) \overset{c}{\approx} \text{EXEC}_{\pi',\mathcal{A}}(\vec{x}).$$

*If $\mathcal{S}$ is restricted to black-box simulation of real-life adversaries, as described above, then we say that π t-*securely computes f in a simulatable way *in the computational scenario.*

B.4 Non-committing encryption

We present a concise definition of a non-committing encryption protocol in our multi-party scenario. First define the bit transmission function $\text{BTR} : \{0,1,\perp\}^n \to \{0,1,\perp\}^n$. This function is parameterized by two identities of parties (i.e., indices $s, r \in [n]$), with the following interpretation. $\text{BTR}_{s,r}$ describes the secure transmission of a bit from party P_s (the sender) to party P_r (the receiver). That is, for $\vec{x} = x_1, \ldots, x_n \in \{0,1,\perp\}^n$ let

$$\text{BTR}_{s,r}(\vec{x})_i = \begin{cases} x_s & \text{if } i = r \\ \perp & \text{otherwise} \end{cases}$$

where $\text{BTR}_{s,r}(\vec{x})_i$ is the i^{th} component of the vector $\text{BTR}_{s,r}(\vec{x})$. We are interested in input vectors $\vec{x}$ where x_s (i.e., the senders input) is in $\{0,1\}$. All other inputs are assumed to be $\perp$.

Definition B.10 *Let $s, r \in [n]$ and $s \neq r$. A protocol ε is a t-resilient (in the presence of $\mathcal{T}$-semi-honest parties and adaptive adversaries),* non-committing encryption *protocol (from P_s to P_r) if ε t-securely computes $\text{BTR}_{s,r}$, in a simulatable way, in the computational model, in the presence $\mathcal{T}$-semi-honest parties and an adaptive adversary.*

It may not be immediately evident how Definition B.10 corresponds to the informal description of non-committing encryptions, presented in Section 2.3. A closer look, however, will show that the requirements from the simulator associated with a non-committing encryption protocol (according to Definition B.10) imply these informal descriptions. In particular, in the case where the simulated adversary corrupts the sender and receiver only after the last communication round, the simulator has to first generate some simulated communication between the parties, without knowing the transmitted bit. (This communication serves as the "dummy ciphertext".) When the sender and/or the receiver are later corrupted, the simulator has to generate internal data that correspond to any value of the transmitted bit.

C Common-domain trapdoor systems

Definition C.1 *A* common-domain trapdoor system *is an infinite set of finite permutations $\{f_{\alpha,\beta} : D_\alpha \overset{\text{1-1}}{\mapsto} D_a\}_{\alpha,\beta \in P}$, where $P \subseteq \{0,1\}^* \times \{0,1\}^*$, so that*

- domain selection: *There exists a probabilistic polynomial-time algorithm G_1 so that on input 1^n, algorithm G_1 outputs a description $\alpha \in \{0,1\}^n$ of domain D_α.*
- function selection: *There exists a probabilistic polynomial-time algorithm G_2 so that on input α, algorithm G_2 outputs a pair $(\beta, t(\beta))$ so that $(\alpha, \beta) \in P$. (β is a description of a permutation over D_α and $t(\beta)$ is the corresponding trapdoor.)*
- domain sampling: *There exists a probabilistic polynomial-time algorithm S that on input α, uniformly selects an element of D_a.*
- function evaluation: *There exists a polynomial-time F that on inputs $(\alpha, \beta) \in P$ and $x \in D_\alpha$ returns $f_{\alpha,\beta}(x)$.*
- function inversion: *There exists a polynomial-time I that on inputs $(\alpha, t(\beta))$ and $y \in D_\alpha$, where $(\alpha, \beta) \in P$, returns $f^{-1}_{\alpha,\beta}(y)$.*
- one-wayness: *For any probabilistic polynomial-time algorithm A, the probability that on input $(\alpha, \beta) \in P$ and $y = f_{\alpha,\beta}(x)$, algorithm A outputs x is negligible (in n), where the probability distribution is over the random choices of $\alpha = G_1(1^n)$, $\beta = G_2(\alpha)$, $x = S(\alpha)$ and the coin tosses of algorithm A.*

The standard definition of trapdoor permutations can be derived from the above by replacing the two selection algorithms, G_1 and G_2, by a single algorithm G that on input 1^n generates a pair $(\beta, t(\beta))$ so that β specifies a domain D_β as well as a permutation f_β over this domain (and $t(\beta)$ is f_β's trapdoor). Thus, the standard definition does not guarantee any structural resemblance among domains of different permutations. Furthermore, it does not allow to generate a new permutation with corresponding trapdoor for a given domain (or given permutation). Nevertheless some popular trapdoor permutations can be formulated in a way which essentially meets the requirements of a common-domain trapdoor system. We describe a specific implementation based on the RSA.

A common-domain trapdoor system based on the RSA. On input 1^n, algorithm G_1 outputs $\alpha \in \{0,1\}^n$ which represents an integer, $i(\alpha)$, uniformly selected in $[2^{n/2}]$. On input α, algorithm G_2 uniformly selects two primes, P and Q, so that their product, N, resides in the interval $[2^n + i(\alpha) \cdot 2^{n/2}, 2^n + (i(\alpha)+1) \cdot 2^{n/2}]$ and a pair of integers (e, d) so that $e \cdot d \equiv 1 \pmod{\phi(N)}$, where $\phi(N) = (P-1) \cdot (Q-1)$. The corresponding domain D_α equals $[2^n + i(\alpha) \cdot 2^{n/2}]$, which is a subset containing all but a negligible fraction of Z_N. The permutation and its trapdoor are defined as usual (i.e., $\beta = (N, e), t(\beta) = (N, d)$ and $f_{\alpha,\beta}(x) \stackrel{\text{def}}{=} x^e \bmod N$). Indeed this is not a permutation over D_α, but rather a 1-1 mapping of D_α to some set R_α, however the symmetric difference between D_α and R_α is negligible and can be ignored. In particular, its effect on our constructions is negligible.

An alternative construction of common-domain trapdoor system. Here we use an arbitrary family of trapdoor permutations, $\{f_\beta; D_\beta \overset{\text{1-1}}{\mapsto} D_\beta\}$, with the extra property that the domain of any permutation, generated on input 1^n, has non-negligible density inside $\{0,1\}^n$ (i.e., $|D_\beta| \geq \frac{1}{\text{poly}(|\beta|)} \cdot 2^{|\beta|}$). We construct a common-domain family where the domain is $\{0,1\}^n$ and the permutations are natural extentions of the given permutations. That is, we let $G_1(1^n) = 1^n$,

$G_2(1^n) = G(1^n)$ and extend f_β into g_β so that $g_\beta(x) = f_\beta(x)$ if $x \in D_\beta$ and $g_\beta(x) = x$ otherwise. This yields a collection of "common-domain" permutations, $\{g_\beta; \{0,1\}^{|\beta|} \stackrel{1\text{-}1}{\mapsto} \{0,1\}^{|\beta|}\}$, which are weakly one-way. Employing the construction of [GILVZ] we obtain a proper common-domain system (of comparable security).

D Proofs

D.1 Sketch of proof of Theorem 3.2

Let π' be a non-erasing protocol for π and let $\mathcal{S}$ be a simulator for π' *in the secure channels setting*. For simplicity we assume that in protocol π, as well as in the interaction generated by $\mathcal{S}$, each party sends on bit to each other party in each round. Let δ be the (computational-model) simulator that corresponds to the non-erasing protocol ε' for the non-committing encryption protocol ε. Given these two different simulators, we construct a simulator $\tilde{\mathcal{S}}$ for protocol $\tilde{\pi}$ in the computational setting. The simulator $\tilde{\mathcal{S}}$ will be a modification of $\mathcal{S}$ and will use several copies of δ as subroutines.

Recall that $\mathcal{S}$ is supposed to interact with a black-box representing a real-life adversary *in the secure channels setting*. That is, at each round $\mathcal{S}$ generates all the messages sent from uncorrupted parties to corrupted parties. Furthermore, whenever the black-box decides to corrupt some party P, machine $\mathcal{S}$ generates internal data for P which is consistent with P's input and with the messages previously sent by P to corrupted parties.

The simulator $\tilde{\mathcal{S}}$, interacts with a black box representing an arbitrary real-life adversary *in the computational setting*, denoted $\tilde{\mathcal{A}}$. The simulator $\tilde{\mathcal{S}}$ is identical to $\mathcal{S}$ with the exception that for each bit sent in the interaction simulated by $\mathcal{S}$, the simulator $\tilde{\mathcal{S}}$ invokes a copy of δ and $\tilde{\mathcal{S}}$ incorporates the outputs of the various copies of δ in its (i.e., $\tilde{\mathcal{S}}$'s) communication with $\tilde{\mathcal{A}}$. Likewise, $\tilde{\mathcal{S}}$ extracts the transmitted bits from the invocations of δ corresponding to message transmissions from corrupted parties to uncoruppted ones. (The way $\tilde{\mathcal{S}}$ handles these invocation will be discussed below.) At this point we stress that $\tilde{\mathcal{A}}$ is the only adversary that $\tilde{\mathcal{S}}$ needs to simulate and to this end it "emulates" real-life adversaries of its choice for the copies of δ. In particular, when $\mathcal{S}$ asks to corrupt some party P, the simulator $\tilde{\mathcal{S}}$ corrupts the same party P. When $\mathcal{S}$ generates P's view *in the secure channel setting*, $\tilde{\mathcal{S}}$ will complete this view into P's view *in the computational setting* by using the various copies of δ.

We describe how $\tilde{\mathcal{S}}$ handles the various copies of δ. As stated above, $\tilde{\mathcal{S}}$ emulates a real-life adversary for each copy of δ using the communication tapes by which this copy is supposed to interact with its black-box/adversary. The information that δ expects to receive form its black box is extracted, in the obvious manner, from the information that $\tilde{\mathcal{S}}$ receives from $\tilde{\mathcal{A}}$. That is, $\tilde{\mathcal{S}}$ hands δ the messages, sent by the corrupted parties, that are relevant to the corresponding invocation of ε'. Furthermore, all the past and current requests for corrupting parties (issued by $\tilde{\mathcal{A}}$) are handed over to δ. The partial view received from each copy of δ is used in the emulation of the corresponding black-box (of this δ-copy) as well as incorporated in the information handed by $\tilde{\mathcal{S}}$ to $\tilde{\mathcal{A}}$. When $\tilde{\mathcal{A}}$ asks to corrupt some party P, the simulator $\tilde{\mathcal{S}}$ emulates a 'corrupt P' request to each copy of δ and obtains the internal data of P in the corresponding sub-protocol ε which it (i.e., $\tilde{\mathcal{S}}$) hands to $\tilde{\mathcal{A}}$ (along with the information obtained by $\mathcal{S}$ – the secure channel simulator). Finally, observe that $\delta = \delta_{s,r}$ (where P_s and P_r are the designated sender and receiver) also expects to interact with parties in the ideal-model. This interaction consists of issuing 'corrupt' requests and obtaining the internal data (of the ideal model). This interaction is (also) emulated by $\tilde{\mathcal{S}}$ as follows. Whenever δ wishes to corrupt a party P which is either P_s or P_r, the simulator $\tilde{\mathcal{S}}$ finds out which bit, σ, was supposed to be sent in this invocation of $\varepsilon'_{r,s}$ and passes σ to $\delta_{r,s}$. We stress that σ is available to $\tilde{\mathcal{S}}$ since at this point in time P has already been corrupted and furthermore $\tilde{\mathcal{S}}$ (which mimics $\mathcal{S}$) has already obtained P's view in the secure channel setting. (Here we use Definitions B.9 and B.10 which guarantee that δ corrupts a party only if this party is already corrupted by δ's black box. We also use the fact that $\tilde{\mathcal{S}}$ is playing δ's black box and is issuing a 'corrupt P' request only after receiving such a request from $\tilde{\mathcal{A}}$ and having simulated this corruption as $\mathcal{S}$.) In case P is neither P_s not P_r the simulator $\tilde{\mathcal{S}}$ passes $\perp$ (as P's input) to δ.

Let $\tilde{\pi}'$ be a non-erasing protocol for $\tilde{\pi}$ and $\tilde{\mathcal{A}}$ be as above (i.e., an arbitrary real-life adversary in the computational setting). We claim that $\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}$ (i.e., the ideal-model adversary $\tilde{\mathcal{S}}$ with black-box access to $\tilde{\mathcal{A}}$) properly simulates the execution of $\tilde{\pi}'$. We need to show that for any adversary $\tilde{\mathcal{A}}$ and for any input $\vec{x}$ we have

$$\text{IDEAL}_{f,\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x}).$$

Here we present only a rough sketch of the proof of this claim. The plan is to construct a real-life adversary $\mathcal{A}$ in the secure channels setting, and prove the following sequence of equalities by which the above claim follows:

$$\text{IDEAL}_{f,\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}}(\vec{x}) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}}}(\vec{x}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi',\mathcal{A}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x}) \quad (2)$$

Regardless of what $\mathcal{A}$ is, the second equality follows immediately from the hypothesis that $\mathcal{S}$ is a simulator for π' (the non-erasing protocol for π) *in the secure channels setting*. It remains to construct $\mathcal{A}$ so that the other two equalities hold.

The real-life adversary $\mathcal{A}$ of the secure channel setting will operate via a simulation of $\tilde{\mathcal{A}}$ (the real-life adversary of the computational setting), imitating the simulation carried out by $\tilde{\mathcal{S}}$. That is, for each bit communicated by π, machine $\mathcal{A}$ will invoke a copy of δ while emulating an adversary in accordance with $\tilde{\mathcal{A}}$. In particular, $\tilde{\mathcal{A}}$ will be given all ciphertexts sent in the open as well as all internal data of corrupted parties (regardless if these parties were corrupted before, during or after the 'real' transmission). Furthermore, when $\tilde{\mathcal{A}}$ corrupts a party P, machine $\mathcal{A}$ corrupts P and hands $\tilde{\mathcal{A}}$ the internal data of P, along with the outputs of the relevant copies δ, just as $\tilde{\mathcal{S}}$ does. At the end of the computation $\mathcal{A}$ outputs whatever $\tilde{\mathcal{A}}$ outputs (that is, $\mathcal{A}$ outputs $\tilde{\mathcal{A}}$'s view of the computation). It follows from the definition of $\mathcal{A}$ that the execution of $\mathcal{S}$, with black-box access to $\mathcal{A}$, is in fact identical to the execution of $\tilde{\mathcal{S}}$ with black-box access to $\tilde{\mathcal{A}}$. Thus, $\text{IDEAL}_{f,\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}}(\vec{x}) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}}}(\vec{x})$ which establishes the first equality in Eq. (2).

It remains to show that $\text{EXEC}_{\pi',\mathcal{A}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x})$. Essentially the difference between these two executions is that $\text{EXEC}_{\pi',\mathcal{A}}(\vec{x})$ is a real-life execution in the secure channel setting which is augmented by invocations of δ (performed by $\mathcal{A}$), whereas $\text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x})$ is a real-life execution in the computational setting in which honest parties use the encryption protocol ε'. However, the security of ε means that invocations of δ are indistinguishable from executions by ε' (both in presence of adaptive adversaries). Using induction on the number of rounds, one thus establishes the last equality of Eq. (2).

D.2 Proof of Claim 3.3

Assume for contradiction that the claim does not hold. Then, without loss of generality, there exists a positive polynomial p so that for infinitely many n's, we have

$$\text{Prob}\left(|\{y \in D_\alpha : B(f_b^{-1}(y)) = B(f_a^{-1}(y))\}| > (\frac{1}{2} + \frac{1}{p(n)}) \cdot |D_\alpha|\right) > \frac{1}{p(n)}$$

when f_a and f_b are independently generated from $\alpha = G_1(1^n)$. This means that for these (α, a, b)'s $B(f_a^{-1}(y))$ gives a non-trivial prediction for $B(f_b^{-1}(y))$. Intuitively this cannot be the case and indeed this lead to contradiction as follows.

Given $a = (\alpha, \beta) \in P$ and $y \in D_\alpha$ we may predict $B(f_a^{-1}(y))$ as follows. First we randomly generate a new permutation. f_b, over D_α, together with its trapdoor. Next we test to see if indeed $B(f_a^{-1}(z))$ is correlated with $B(f_b^{-1}(z))$. (The testing is done by uniformly selecting polynomially many x_i's in D_α, computing $z_i = f_a(x_i)$, and comparing $B(f_a^{-1}(z_i)) = B(x_i)$ with $B(f_b^{-1}(z_i))$.) If a non-negligible correlation is detected then we output $B(f_b^{-1}(y)$ (as our prediction for $B(f_a^{-1}(y))$). Otherwise we output a uniformly selected bit. (Note that $|\text{Prob}(B(x) = 1) - \frac{1}{2}|$ must be negligible otherwise a constant function contradicts the hard-core hypothesis.)

D.3 Proof of Lemma 3.4

Assume without loss of generality that the private key is f_a^{-1}. Then, the receiver outputs the majority value of the bits $B(f_a^{-1}(y_1)), \ldots, B(f_a^{-1}(y_{16k}))$. Recall that $8k$ of the y_i's are associated with f_a. Out of them, $5k$ (of the y_i's) satisfy $B(f_a^{-1}(y_i)) = B(x_i) = \sigma$, and $3k$ satisfy $B(f_a^{-1}(y_i)) = B(x_i) = 1 - \sigma$. Thus, the receiver outputs $1 - \sigma$ only if at least $5k$ out of the rest of the y_i's (that is, the y_i's associated with f_b) satisfy $B(f_a^{-1}(y_i)) = 1 - \sigma$. However, Eq. (1) implies that $|\text{Prob}(B(f_a^{-1}(y_i) = \sigma) - \frac{1}{2}|$ is negligible for each y_i associated with f_b. Thus only an expected $4k$ of the y_i's associated with f_b satisfy $B(f_a^{-1}(y_i)) = 1 - \sigma$. Using a large deviation bound, it follows that decryption errors occur with probability $2^{-\Omega(k)}$.

D.4 Proof of Lemma 3.5

Let us first prove a simple technical claim, that will help us in proving Lemma 3.5. Let BIN_m denote the binomial distribution over $[m]$.

Claim D.1 *There exists an efficiently samplable distribution μ over $\{0, 1, ..., 4k\}$ so that the distribution $\tilde{\mu}$ constructed by sampling an integer from μ and adding $2k$ is statistically close to BIN_{8k}. That is, the statistical distance between $\tilde{\mu}$ and BIN_{8k} is $2^{-\Omega(k)}$.*

Proof: Let $\text{BIN}_{8k}(i)$ denote the probability of i under BIN_{8k} (i.e., $\text{BIN}_{8k}(i) = \binom{8k}{i} \cdot 2^{-8k}$). We construct the distribution μ (over $\{0, 1, ..., 4k\}$) so that Prob$(\mu = i) = \text{BIN}_{8k}(i + 2k)$ for $i = 1, ..., 4n$ and Prob$(\mu = 0)$ equals the remaining mass of BIN_{8k} (i.e., it equals $\sum_{i=0}^{2k} \text{BIN}_{8k}(i) + \sum_{i=6k+1}^{8k} \text{BIN}_{8k}(i)$).

It can be easily seen that each $i \in \{2k + 1, ..., 6k\}$ occurs under $\tilde{\mu}$ with exactly the same probability as under BIN_{8k}. Integers i such that $i < 2k$ or $i > 6k$ have probability 0 under $\tilde{\mu}$ (whereas $2k$ is more likely to occur under $\tilde{\mu}$ than under BIN_{8k}). Thus, the statistical distance between $\tilde{\mu}$ and BIN_{8k} equals the probability, under BIN_{8k}, that i is smaller than $2k$ or larger than $6k$. This probability is bounded by $2^{-\Omega(k)}$. $\square$

Before describing how the dummy ciphertext $\vec{z}$ and the rest of the data are constructed, we summarize, in Figure 2, the distribution of the hard-core bits, $B(f_a^{-1}(y_1)), ..., B(f_a^{-1}(y_{16k}))$ and $B(f_b^{-1}(y_1)), ..., B(f_b^{-1}(y_{16k}))$, with respect to a real encryption $y_{\phi(1)}, \ldots, y_{\phi(16k)}$ of the bit $\sigma = 0$. Here $\tilde{\text{B}}\tilde{\text{I}}\tilde{\text{N}}_{8k}$ denotes the distribution of the number of '1's in $B(f_b^{-1}(y_i))$ for $i = 1, ..., 8k$. Eq. (1) implies that the statistical difference between BIN_{8k} and $\tilde{\text{B}}\tilde{\text{I}}\tilde{\text{N}}_{8k}$ is negligible. The distribution of $B(f_a^{-1}(y_i))$ for $i = 8k + 1, ..., 16k$ is similar. Given only λ_0 (or only λ_1), only three-quarters

	$I = \{1, ..., 8k\}$	$I = \{8k + 1, ..., 16k\}$
$\forall i \in I$	$y_i = f_a(x_i)$	$y_i = f_b(x_i)$
$\sum_{i \in I} B(f_a^{-1}(y_i)) =$	$3k$	$\tilde{\text{B}}\tilde{\text{I}}\tilde{\text{N}}_{8k}$
$\sum_{i \in I} B(f_b^{-1}(y_i)) =$	$\tilde{\text{B}}\tilde{\text{I}}\tilde{\text{N}}_{8k}$	$3k$

Figure 2: The distribution of the $B(f_s^{-1}(y_i))$'s with respect to λ_0, where $s \in \{a, b\}$. (The case of λ_1 is similar, with the exception that 5k is replaced for 3k.)

of the $B(f_s^{-1}(y_i))$'s, $i \in [16k]$ and $s \in \{a, b\}$, are known. Specifically, consider $\lambda_\sigma = (\sigma; \vec{x}, \phi; \vec{y}; r, f_r^{-1})$, and suppose that $r = a$. Then all the $B(f_a^{-1}(y_i))$'s can be computed using f_a^{-1}. In addition, for $i = 8k + 1, ..., 16k$, $B(f_b^{-1}(y_i)) = B(x_i)$ is known too. However, for $i \in [8k]$, $B(f_b^{-1}(y_i)) = B(f_b^{-1} f_a(x_i)))$ is not known and in fact it is (computationally) unpredictable (from λ_σ). A similar analysis holds for $r = b$; in this case the unpredictable bits are $B(f_a^{-1}(y_i)) = B(f_a^{-1} f_b(x_i)))$ for $i = 8k + 1, ..., 16k$.

INITIAL CONSTRUCTION AND CONDITIONS: Keeping the structure of λ_σ in mind, we construct $\vec{z}$, along with $\vec{x}^{(0)}$, $\vec{x}^{(1)}$, $\phi^{(0)}$, $\phi^{(1)}$, $r^{(0)}$ and $r^{(1)}$, as follows. First, we select uniformly a bijection, ρ, of $\{0,1\}$ to $\{a,b\}$ (i.e., either $\rho(0)=a$ and $\rho(1)=b$ or the other way around) and set $r^{(0)}=\rho(0)$ and $r^{(1)}=\rho(1)$. Next, we choose, in the way described below, two binary vectors $\vec{\gamma}^{(0)}=\gamma_1^{(0)},\ldots,\gamma_{16k}^{(0)}$ and $\vec{\gamma}^{(1)}=\gamma_1^{(1)},\ldots,\gamma_{16k}^{(1)}$. We choose random values $v_1,\ldots,v_{16k}$ such that $\gamma_i^{(0)}=B(f_{\rho(0)}^{-1}(v_i))$ and $\gamma_i^{(1)}=B(f_{\rho(1)}^{-1}(v_i))$, for each $i\in[16k]$. We uniformly select a permutation ψ over $[16k]$ and let the permuted vector $v_{\psi(1)},\ldots,v_{\psi(16k)}$ be the dummy ciphertext $\vec{z}=(z_1,...,z_{16k})$. It remains to determine $\phi^{(0)}$ and $\phi^{(1)}$, which in turn determine $\vec{x}^{(0)}$ and $\vec{x}^{(1)}$ so that $x_i^{(\sigma)}=f_a^{-1}(z_{(\phi^{(\sigma)})^{-1}(i)})$ for $i\in[8k]$ and $x_i^{(\sigma)}=f_b^{-1}(z_{\phi^{(\sigma)}(i)})$ otherwise. This should be done so that both permutations $\phi^{(0)}$ and $\phi^{(1)}$ are uniformly (but not necessarily independently) distributed and so that the known $B(f_s^{-1}(y_i^{(\sigma)}))$'s match the distribution seen in a legitimate encryption of σ. We stress that $(\sigma;\vec{x}^{(\sigma)},\phi^{(\sigma)};\vec{z};r^{(\sigma)},f_{r(\sigma)}^{-1})$ should appear as a valid encryption of σ. In particular, for each $\sigma\in\{0,1\}$ there should exist a permutation $\psi^{(\sigma)}$ $(=(\phi^{(\sigma)})^{-1}\circ\phi)$ over $[16k]$ so that[12]

1. $\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(a))}=B(f_a^{-1}(v_{\psi^{(\sigma)}(i)}))=B(f_a^{-1}(z_{\phi^{(\sigma)}(i)}))=B(x_i^{(\sigma)})=\sigma$, for $i=1,...,5k$.
   (E.g., if $\rho(0)=a$ this means $\gamma_{\psi^{(\sigma)}(i)}^{(0)}=\sigma$.)
2. $\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(a))}=B(f_a^{-1}(v_{\psi^{(\sigma)}(i)}))=B(f_a^{-1}(z_{\phi^{(\sigma)}(i)}))=B(x_i^{(\sigma)})=1-\sigma$, for $i=5k+1,...,8k$.
   (E.g., if $\rho(0)=a$ this means $\gamma_{\psi^{(\sigma)}(i)}^{(0)}=1-\sigma$.)
3. $\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(b))}=B(f_b^{-1}(v_{\psi^{(\sigma)}(i)}))=B(f_b^{-1}(z_{\phi^{(\sigma)}(i)}))=B(x_i^{(\sigma)})=\sigma$, for $i=8k+1,...,13k$.
   (E.g., if $\rho(0)=a$ this means $\gamma_{\psi^{(\sigma)}(i)}^{(1)}=\sigma$.)
4. $\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(b))}=B(f_b^{-1}(v_{\psi^{(\sigma)}(i)}))=B(f_b^{-1}(z_{\phi^{(\sigma)}(i)}))=B(x_i^{(\sigma)})=1-\sigma$, for $i=13k+1,...,16k$.
   (E.g., if $\rho(0)=a$ this means $\gamma_{\psi^{(\sigma)}(i)}^{(1)}=1-\sigma$.)
5. Let $I=[8k]$ if $\rho(\sigma)=b$ and $I=\{8k+1,...,16k\}$ otherwise. Then, $\gamma_{\psi^{(\sigma)}(i)}^{(\sigma)}=B(f_{\rho(\sigma)}^{-1}(v_{\psi^{(\sigma)}(i)}))=B(f_{\rho(\sigma)}^{-1}(z_{\phi^{(\sigma)}(i)}))=B(f_{\rho(\sigma)}^{-1}(f_{\rho(1-\sigma)}(x_i^{(\sigma)})))$ equals σ with probability negligibly close to $\frac{1}{2}$, for $i\in I$.
   (E.g., for $\rho(0)=a$ and $\sigma=0$ we have Prob$(\gamma_{\psi^{(\sigma)}(i)}^{(0)}=1)\approx\frac{1}{2}$ for $i=8k+1,...,16k$, whereas for $\rho(0)=a$ and $\sigma=1$ we have Prob$(\gamma_{\psi^{(\sigma)}(i)}^{(1)}=1)\approx\frac{1}{2}$ for $i=1,...,8k$.)

This allows setting $\phi^{(\sigma)}=\psi\circ(\psi^{(\sigma)})^{-1}$ so that $x_{\phi^{(\sigma)}(i)}^{(\sigma)}$ is "mapped" to z_i while $\phi^{(\sigma)}$ is uniformly distributed (i.e., $x_i^{(\sigma)}=f_a^{-1}(v_{\psi^{(\sigma)}(i)})=f_a^{-1}(z_{\psi^{-1}(\phi^{(\sigma)}(i))})=f_a^{-1}(z_{(\phi^{(\sigma)})^{-1}(i)})$ for $i\in[8k]$ and $x_i^{(\sigma)}=f_b^{-1}(z_{\phi^{(\sigma)}(i)})$ otherwise).

INITIAL SETTING OF $\vec{\gamma}^{(0)}$, $\vec{\gamma}^{(1)}$, $\psi^{(0)}$ AND $\psi^{(1)}$: The key issue is how to select $\vec{\gamma}^{(0)}$ and $\vec{\gamma}^{(1)}$ so that the five condition stated above hold (for both $\sigma=0$ and $\sigma=1$). As a first step towards this goal we consider the four sums

$$S_1^\sigma \stackrel{\rm def}{=} \sum_{i=1}^{8k}\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(a))}\ ,\ \ S_2^\sigma \stackrel{\rm def}{=} \sum_{i=8k+1}^{16k}\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(b))}\ ,\ \ S_3^\sigma \stackrel{\rm def}{=} \sum_{i=1}^{8k}\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(b))}\ ,\ \ S_4^\sigma \stackrel{\rm def}{=} \sum_{i=8k+1}^{16k}\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(a))}$$

The above conditions imply $S_1^\sigma=S_2^\sigma=5k\cdot\sigma+3k\cdot(1-\sigma)=3k+2k\sigma$ as well as $S_3^\sigma\stackrel{\rm d}{=}\widetilde{\rm BIN}_{8k}$ if $\rho(\sigma)=b$ and $S_4^\sigma\stackrel{\rm d}{=}\widetilde{\rm BIN}_{8k}$ otherwise. (Note that S_3^σ, S_4^σ and $\widetilde{\rm BIN}_{8k}$ are random variables.)

[12] In each of the following five conditions, the first equality is by the construction of the v_i's, the second equality is by the definition of the z_i's, and the third equality represents the relation between $\vec{x}^{(\sigma)}$, $\vec{z}$ and $\phi^{(\sigma)}$ that holds in a valid encryption (of σ). In conditions (1) through (4), the last equality represents the relation between $\vec{x}^{(\sigma)}$ and σ that holds in a valid encryption of σ. In condition (5), the last equality represents the information computable from $\vec{z}$ using (the trapdoor) $f_{r(\sigma)}^{-1}$. Here we refer to the inverses of the z_i's which are not $x_i^{(\sigma)}$'s. The hard-core value of these inverses should be uniformly distributed.

To satisfy the above summation conditions we partition $[16k]$ into 4 equal sized subsets denoted I_1, I_2, I_3, I_4 (e.g., $I_1 = [4k], I_2 = \{4k+1, ..., 8k\}, I_3 = \{8k+1, ..., 12k\}$ and $I_4 = \{12k+1, ..., 16k\}$). This partition induces a similar partition on the $\gamma_i^{(0)}$'s and the $\gamma_i^{(1)}$'s. The $\gamma_i^{(0)}$'s and the $\gamma_i^{(1)}$'s in each set are chosen using four different distributions which satisfy the conditions summarized in Figure 3. Suppose $\rho(0) = a$. Then, we may set $\psi^{(0)}([8k]) = I_1 \cup I_2$

	$I = I_1$	$I = I_2$	$I = I_3$	$I = I_4$
$\sum_{i \in I} \gamma_i^{(0)} \stackrel{\rm d}{=}$	$3k$	0	$2k$	μ
$\sum_{i \in I} \gamma_i^{(1)} \stackrel{\rm d}{=}$	μ	$4k$	$2k$	k

Figure 3: The distribution of the $\gamma^{(0)}$'s and $\gamma^{(1)}$'s. (μ is as in Claim D.1.)

and $\psi^{(0)}(\{8k+1, ..., 16k\}) = I_3 \cup I_4$, and $\psi^{(1)}([8k]) = I_1 \cup I_3$ and $\psi^{(1)}(\{8k+1, ..., 16k\}) = I_2 \cup I_4$, where $\pi(I) = J$ means that the permutation π maps the elements of the set I onto the set J. (It would have been more natural but less convenient to write $(\psi^{(1)})^{-1}(I_1 \cup I_3) = [8k]$ and $(\psi^{(1)})^{-1}(I_2 \cup I_4) = \{8k+1, 16k\}$.) We claim that, for each $\sigma \in \{0, 1\}$, the above setting satisfies the three relevant summation conditions. Consider, for example, the case $\sigma = 0$ (depicted in Figure 4). Then, $S_1^0 = \sum_{i=1}^{8k} \gamma_i^{(0)} = 3k$ and $S_2^0 = \sum_{i=8k+1}^{16k} \gamma_i^{(1)} = 3k$ as required. Considering

	$I = \{1, ..., 8k\} = (\psi^{(0)})^{-1}(I_1 \cup I_2)$	$I = \{8k+1, ..., 16k\} = (\psi^{(0)})^{-1}(I_3 \cup I_4)$
$\sum_{i \in I} \gamma_i^{(0)} =$	$S_1^0 = 3k + 0 = 3k$	$S_4^0 = 2k + \mu \stackrel{\rm d}{=} \tilde{\rm B}\tilde{\rm I}\tilde{\rm N}_{8k}$
$\sum_{i \in I} \gamma_i^{(1)} =$	no condition	$S_2^0 = 2k + k = 3k$

Figure 4: Using $\psi^{(0)}$ the $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s satisfy the summation conditions S_1^0, S_2^0 and S_4^0.

$S_4^0 = \sum_{i=8k+1}^{16k} \gamma_i^{(0)}$ we observe that it is distributed as $2k + \mu = \tilde{\mu}$ (of Claim D.1) which in turn is statistically close to $\tilde{\rm B}\tilde{\rm I}\tilde{\rm N}_{8k}$. We stress that the above argument holds for any way of setting the $\psi^{(\cdot)}$'s as long as they obey the equalities specified (e.g., for any bijection $\pi : I_1 \cup I_2 \stackrel{\text{1-1}}{\mapsto} I_1 \cup I_3$, we are allowed to set $\psi^{(1)}(i) = \pi(i)$ for all $i \in I_1 \cup I_2$). The case $\sigma = 1$ follows similarly; here $S_1^1 = \sum_{i \in I_1 \cup I_3} \gamma_i^{(0)} = 5k$, $S_2^1 = \sum_{i \in I_2 \cup I_4} \gamma_i^{(1)} = 5k$ and $S_3^1 = \sum_{i \in I_1 \cup I_3} \gamma_i^{(1)} = \mu + 2k$ (see Figure 5). In case $\rho(0) = b$ we set $\psi^{(0)}([8k]) = I_3 \cup I_4$, $\psi^{(0)}(\{8k+1, ..., 16k\}) = I_1 \cup I_2$, $\psi^{(1)}([8k]) = I_2 \cup I_4$

	$I = \{1, ..., 8k\} = (\psi^{(1)})^{-1}(I_1 \cup I_3)$	$I = \{8k+1, ..., 16k\} = (\psi^{(1)})^{-1}(I_2 \cup I_4)$
$\sum_{i \in I} \gamma_i^{(0)} =$	$S_1^1 = 3k + 2k = 5k$	no condition
$\sum_{i \in I} \gamma_i^{(1)} =$	$S_3^1 = \mu + 2k \stackrel{\rm d}{=} \tilde{\rm B}\tilde{\rm I}\tilde{\rm N}_{8k}$	$S_2^1 = 4k + k = 5k$

Figure 5: Using $\psi^{(1)}$ the $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s satisfy the summation conditions S_1^1, S_2^1 and S_3^1.

and $\psi^{(1)}(\{8k+1, ..., 16k\}) = I_1 \cup I_3$. The claim that, for each $\sigma \in \{0, 1\}$, the above setting satisfies the three relevant summation conditions, is shown analogously.

REFINEMENT OF $\vec{\gamma}^{(0)}$, $\vec{\gamma}^{(1)}$, $\psi^{(0)}$ AND $\psi^{(1)}$: However, the above summation conditions do not guarantee satisfaction of all the five conditions. In particular, we must use permutations $\psi^{(\cdot)}$ which guarantee the correct positioning visible bits within the $8k$-bit long block. That is, we must have

$$\begin{aligned} (\gamma_{\psi^{(\sigma)}(1)}^{(\rho^{-1}(a))}, ..., \gamma_{\psi^{(\sigma)}(8k)}^{(\rho^{-1}(a))}) &= (\sigma^{5k}, (1-\sigma)^{3k}) \\ (\gamma_{\psi^{(\sigma)}(8k+1)}^{(\rho^{-1}(a))}, ..., \gamma_{\psi^{(\sigma)}(16k)}^{(\rho^{-1}(a))}) &= (\sigma^{5k}, (1-\sigma)^{3k}) \end{aligned}$$

that is, equality between the sequences and not merely equality in the number of 1's. Clearly there is no problem to set the $\psi^{(\cdot)}$'s so that these equalities hold and thus Conditions (1) through (4) are satisfied. It is left to satisfy Condition (5).

Suppose that $\rho(\sigma) = a$. In this case the third summation requirement guarantees $\sum_{i=8k+1}^{16k} \gamma^{(\sigma)}_{\psi^{(\sigma)}(i)} \stackrel{\mathrm{d}}{=} \tilde{\tilde{\tilde{\mathrm{BIN}}}}_{8k}$. This is indeed consistent with the requirement that these $\gamma^{(\sigma)}_{\psi^{(\sigma)}(i)}$'s are almost uniformly and independently distributed. But this is not sufficient. In particular, we also need $\sum_{i\in J} \gamma^{(\sigma)}_{\psi^{(\sigma)}(i)} \stackrel{\mathrm{d}}{=} \tilde{\tilde{\tilde{\mathrm{BIN}}}}_{3k}$, where $J = \{8k < i \leq 16k : \gamma^{(1-\sigma)}_{\psi^{(\sigma)}(i)} = 1-\sigma\}$ and furthermore the above sum needs to be independent of $\sum_{i\in\{8k+1,\ldots,16k\}-J} \gamma^{(\sigma)}_{\psi^{(\sigma)}(i)}$ (which in turn should be statistically close to BIN_{5k}). Let us start with the case $\sigma = 0$. In this case we need

$$\sum_{i\in J} \gamma_i^{(0)} \stackrel{\mathrm{d}}{=} \tilde{\tilde{\tilde{\mathrm{BIN}}}}_{3k}, \tag{3}$$

where $J = \{i \in I_3 \cup I_4 : \gamma_i^{(1)} = 1\}$, and this sum needs to be independent of $\sum_{i\in I_3\cup I_4 - J} \gamma_i^{(0)}$. By Figure 3 we have $|J \cap I_3| = 2k$. We further restrict the distributions $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s so that in part I_3 the four possible outcomes of the pairs $(\gamma_i^{(0)}, \gamma_i^{(1)})$ are equally likely (e.g., for exactly k integers $i \in I_3$ we have $(\gamma_i^{(0)}, \gamma_i^{(1)}) = (0,0)$). Consider $J' = J \cap I_4$ (note $|J'| = k$). To satisfy Eq. (3) we construct a random variable $\mu' \in \{0, 1, ..., k\}$ (analogously to Claim D.1) so that $p_j \stackrel{\mathrm{def}}{=} \mathrm{Prob}(\mu' = j) = \mathrm{BIN}_{3k}(k + j)$ for $j \in [k]$ (with the rest of the mass on $\mu' = 0$) and constrain the $\gamma_i^{(0)}$'s to satisfy $\mathrm{Prob}(\sum_{i\in J'} \gamma_i^{(0)} = j) = p_j$. We get $\sum_{i\in J} \gamma_i^{(0)} = k + \mu' \stackrel{\mathrm{d}}{=} \tilde{\tilde{\tilde{\mathrm{BIN}}}}_{3k}$ (analogously to Claim D.1). A minor problem occurs: the new restriction on the $\gamma_i^{(0)}$'s conditions $\sum_{i\in I_4 - J'} \gamma_i^{(0)}$ which we want to be distributed as some $\mu'' \stackrel{\mathrm{d}}{=} \mathrm{BIN}_{5k} - 2k$ and independently of μ' (the reason being that $\mu' + \mu''$ should be distributed equally to μ). However this condition has a negligible effect since we can sample μ' and μ and set the $\gamma_i^{(0)}$'s accordingly, getting into trouble only in case $\mu < \mu'$ which happens with negligible probability (since $\mathrm{Prob}(\mu < \mu') < \mathrm{Prob}(\mu < k) = 2^{-\Omega(k)}$).

The case $\sigma = 1$ gives rise to the requirement

$$\sum_{i\in J} \gamma_i^{(1)} \stackrel{\mathrm{d}}{=} \tilde{\tilde{\tilde{\mathrm{BIN}}}}_{3k}, \tag{4}$$

where $J = \{i \in I_1 \cup I_3 : \gamma_i^{(0)} = 0\}$, and this sum needs to be independent of $\sum_{i\in I_1\cup I_3 - J} \gamma_i^{(1)}$. To satisfy Eq. (4) we restrict the $\gamma_i^{(1)}$'s in $J' \stackrel{\mathrm{def}}{=} J \cap I_1$ analogously to satisfy $\sum_{i\in J'} \gamma_i^{(1)} = \mu'$. Finally, we observe that generating the $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s at random so that they satisfy the above requirements makes them satisfy Condition (5).

BEYOND THE FIVE CONDITIONS. In the above construction we have explicitly dealt with conditions which obviously have to hold for the construction to be valid. We now show that indeed this suffices. Namely, we claim that

$$(\sigma; \vec{x}^{(\sigma)}, \phi^{(\sigma)}; \vec{z}; r^{(\sigma)}, f^{-1}_{r(\sigma)}) \stackrel{\mathrm{c}}{\approx} \lambda_\sigma = (\sigma; \vec{x}, \phi; \vec{y}; r, f_r^{-1}). \tag{5}$$

Consider the case $\sigma = 0$. Both $r^{(0)}$ and r are uniformly chosen in $\{a, b\}$ and so we consider, w.l.o.g., $r = r^{(0)} = a$. Furthermore, $\phi^{(0)}$ is a random permutation and $f_a(x_i^{(0)}) = z_{\phi^{(0)}}$ for $i = 1, ..., 8k$, and $f_b(x_i^{(0)}) = z_{\phi^{(0)}}$ for $i = 8k+1, ..., 16k$, which matches the situation w.r.t ϕ, $\vec{x}$ and $\vec{y}$. It remains to compare the distributions of $B(f_s^{-1}(\cdot))$'s, $s \in \{a, b\}$, with respect to $\vec{x}^{(0)}$ and with respect to $\vec{x}$. By the above analysis we know that the entries corresponding to $s = a$ and to $(s = b) \wedge (i \leq 8k)$ are distributed similarly in the two cases. Thus, we need to compare $B(f_b^{-1}(f_a(x_1^{(0)}))), ..., B(f_b^{-1}(f_a(x_{8k}^{(0)})))$ and $B(f_b^{-1}(f_a(x_1))), ..., B(f_b^{-1}(f_a(x_{8k})))$. Recall that the x_i's are selected at random subject to $B(x_i) = 0$ for $i = 1, ..., 5k$ and $B(x_i) = 1$ for $i = 5k + 1, ..., 8k$. An analogous condition is imposed on the $x_i^{(0)}$'s but in addition we also have $B(f_b^{-1}(f_a(x_i^{(0)}))) = 1$ for $i = 1, ..., 4k$, and some complicated conditions on $B(f_b^{-1}(f_a(x_i^{(0)}))) = 1$, for $i = 4k + 1, ..., 8k$ (i.e., the distribution of 1's here is governed by μ and furthermore in the first k elements the number of 1's is distributed identically to μ'). Thus, distinguishing $\vec{x}$ from $\vec{x}^{(0)}$ amounts to distinguishing, given $f_a, f_b : D \mapsto D$ and the trapdoor for f_a (but not for f_b), between the two distributions

1. $(u_1, ..., u_{8k})$, where the u_i's are independently selected so that $B(u_i) = 0$ if $i \in [5k]$ and $B(u_i) = 1$ otherwise; and

2. $(w_1, ..., w_{8k})$, where the w_i's are uniformly selected under the conditions
   - $B(w_i) = 0$ if $i \in [5k]$ and $B(u_i) = 1$ otherwise,
   - $B(f_b^{-1}(f_a(w_i))) = 1$ for $i \in [4k]$,
   - $\sum_{i=4k+1}^{5k} B(f_b^{-1}(f_a(w_i))) = \mu'$, and
   - $\sum_{i=5k+1}^{8k} B(f_b^{-1}(f_a(w_i))) = \mu''$, for some $\mu'' \stackrel{\mathrm{d}}{=} \mu - \mu'$.

We claim that distinguishing these two distributions yields a contradiction to the security of the hard-core predicate B. Suppose, on the contrary that an efficient algorithm A can distinguish these two distributions. Using a hybrid argument we construct an algorithm A' which distinguishes the the uniform distribution over $D' \stackrel{\mathrm{def}}{=} \{x \in D : B(x) = \tau\}$ and a distribution over D' that is uniform over both $D'_0 \stackrel{\mathrm{def}}{=} \{x \in D' : B(f_b^{-1}(f_a(x))) = 0\}$ and $D'_1 \stackrel{\mathrm{def}}{=} \{x \in D' : B(f_b^{-1}(f_a(x))) = 1\}$, where τ is a bit which can be efficiently determined. (We stress that the latter distribution is not uniform on D' but rather uniform on each of its two parts.) Without loss of generality, we assume $\tau = 0$. It follows that A' must distinguish inputs uniformly distributed in D'_0 from inputs uniformly distributed in D'_1. We now transform A' into an algorithm, A'', that distinguishes a uniform distribution over $\{y \in D : B(f_b^{-1}(y)) = 0\}$ from a uniform distribution over $\{y \in D : B(f_b^{-1}(y)) = 1\}$. On input $y \in D_\alpha$ and $f_b : D \mapsto D$, algorithm A'' first generates another permutation f_a, over D, together with the trapdoor for f_a. Next, it computes $x = f_a^{-1}(y)$ and stop (outputting 0) if $B(x) = 1$ (i.e., $x \notin D'$). Otherwise, A'', invokes A' on x and outputs $A'(x)$. In this case $B(f_b^{-1}(f_a(x))) = B(f_b^{-1}(y))$ (and $B(x) = 0$) so the output will be significantly different in case $B(f_b^{-1}(y))) = 0$ and in case $B(f_b^{-1}(y))) = 1$. We observe that Prob($B(x) = 0) \approx \frac{1}{2}$ (otherwise a constant function violates the security of B), and conclude that one can a random y with $B(f_b^{-1}(y)) = 0$ from a random y with $B(f_b^{-1}(y)) = 1$ (which contradicts the security of B).

D.5 Sketch of proof of Theorem 3.6

Let P_r be the sender and let P_s be the receiver. Recall that a non-committing encryption protocol is a protocol that securely computes the bit transmission function, $\text{BTR}_{s,r}$, in a simulatable way. Let ε' be a non-erasing protocol for ε. We construct a simulator $\mathcal{S}$ such that $\text{IDEAL}_{\text{BTR}_{s,r},\mathcal{S}^{\mathcal{A}}}(\sigma) \stackrel{\mathrm{d}}{=} \text{EXEC}_{\varepsilon',\mathcal{A}}(\sigma)$, for any $(n-1)$-limited adversary $\mathcal{A}$ and for any input $\sigma \in \{0,1\}$ of P_s.

The simulator $\mathcal{S}$ proceeds as follows. First an invocation of the key generation protocol ε_{G} is simulated, in such a way that $\mathcal{S}$ knows both trapdoors f_a^{-1} and f_b^{-1}. (This can be done using the additional property of the [GMW] Oblivious Transfer protocol, as described above.) For each party P that $\mathcal{A}$ corrupts during this stage, $\mathcal{S}$ hands $\mathcal{A}$ the internal data held by P in the simulated interaction. We stress that as long as at least one party remains uncorrupted, the adversary knows at most *one* of f_a^{-1}, f_b^{-1}. Furthermore, as long as P_r remains uncorrupted, the adversary view of the computation is independent of whether P_r has f_a^{-1} or f_b^{-1}.

Once the simulation of the key generation protocol is completed, $\mathcal{S}$ instructs the trusted party in the ideal model to notify P_r of the function value. (This value is P_s's input, σ.) If at this point either P_s or P_r is corrupted, then $\mathcal{S}$ gets to know the encrypted bit. In this case $\mathcal{S}$ generates a true encryption of the bit σ, according to the protocol. If neither P_s nor P_r are corrupted, then $\mathcal{S}$ generates the values $\vec{z}$, $\vec{x}^{(0)}$, $\vec{x}^{(1)}\phi^{(0)}$, $\phi^{(1)}$, $r^{(0)}$, $r^{(1)}$ as in Lemma 3.5, and lets $\vec{z}$ be the ciphertext that P_s sends to P_r in the simulated interaction.

If at this stage $\mathcal{A}$ corrupts some party P which is not the sender or the receiver, then $\mathcal{S}$ hands $\mathcal{A}$ the internal data held by P in the simulated interaction. If $\mathcal{A}$ corrupts P_s, then $\mathcal{S}$ corrupts P_s in the ideal model and learns σ. Next $\mathcal{S}$ hands $\mathcal{A}$

the values $\vec{x}^{(\sigma)}, \phi^{(\sigma)}$ for P_s's internal data. If $\mathcal{A}$ corrupts P_r, then $\mathcal{S}$ corrupts P_r in the ideal model, learns σ, and hands $\mathcal{A}$ the value $f^{-1}_{r(\sigma)}$ for P_s's internal data.

The validity of the simulation follows from Lemma 3.5 and from the properties of the [GMW] Oblivious Transfer protocol.

E The Oblivious Transfer protocol

The following OT protocol is taken from [GMW].

Oblivious Transfer (OT)

The parties proceed as follows, using a trapdoor-permutations generator and the associated hard-core predicate $B()$.

1. On input $\sigma_1, \sigma_2 \in \{0,1\}$, the sender generates a one-way trapdoor permutation $f : D \rightarrow D$ with its trapdoor f^{-1}, and sends f to the receiver.
2. On input $\tau \in \{1,2\}$, the receiver uniformly selects $x_1, x_2 \in D$, computes $y_\tau = f(x_\tau)$, sets $y_{3-\tau} = x_{3-\tau}$, and sends (y_1, y_2) to the sender.
3. Upon receiving (y_1, y_2), the sender sends the pair $(\sigma_1 \oplus B(f^{-1}(y_1)), \sigma_2 \oplus B(f^{-1}(y_2)))$ to the receiver.
4. Having received (b_1, b_2), the receiver outputs $s_\tau = b_\tau \oplus B(x_\tau)$ (as the message received).

Figure 6: The [GMW] Oblivious Transfer protocol

It can be easily verified that the receiver outputs the correct value of σ_τ in Step 4. Also, if the receiver is semi-honest in the non-erasing sense, then it cannot predict $\sigma_{3-\tau}$ with more than negligible advantage over $\frac{1}{2}$. [13] The sender view of the interaction is uncorrelated with the value of $\tau \in \{1,2\}$. Thus it learns nothing from participating in the protocol.

An important additional property of this protocol is that, in a simulated execution of the protocol, the simulator can learn both σ_1 and σ_2 by uniformly selecting $z_1, z_2 \in D$, and having the receiver R send $f(z_1), f(z_2)$ (in Step 2). Furthermore, if R is later corrupted, then the simulator can "convince" the adversary that R received either σ_1 or σ_2, at wish, by claiming that in Step 2 party R chose either $(x_1, x_2) = (z_1, f(z_2))$ or $(x_1, x_2) = (f(z_1), z_2)$, respectively.

[13] This statement does not hold if R is semi-honest only in the honest-looking sense. Ironically, this 'flaw' is related to the useful (non-committing) feature discussed below.