CHAPTER 3

Adaptively secure computation in the computational setting

We study secure multiparty computation in the presence of adaptive adversaries (see Section 1.3 for an introductory discussion). Although the emphasis of this chapter is on the computational setting, we also sketch the state of affairs in the secure channels setting. We believe that understanding adaptively secure computation in the computational setting is easier if the secure channels setting is first considered.

We first present, in Section 3.1, an overview of the problems encountered when trying to prove adaptive security of protocols, first in the secure channels setting, and then in the computational settings. We also sketch our solution for the computational setting. Next, in Section 3.2 we define a tool, called non-committing encryption, that is central in our solution for the computational setting. In Section 3.3 we present our construction for the case of non-erasing parties. We first show how, given a non-committing encryption scheme, any adaptively secure protocol in the secure channels setting can be transformed into an adaptively secure protocol in the computational setting. Next we describe our construction of a non-committing encryption scheme. In Section 3.4 we suggest a construction for the case of honest-looking parties.

3.1 The problems in proving adaptive security: informal presentation

3.1.1 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.

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 Section 2.3.) 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.

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

3.1.2 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 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.

[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 $\mathcal{S}$.

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-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 $\mathcal{E}^{\text{amb}}$ 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').

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

[3] This "non-committing property" is reminiscent of the "Chameleon blobs" of [Br]. 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.

[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.

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 3.5). Thus, we get

Theorem 3.1 *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.

3.2 Defining 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 3.2 *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 3.2 corresponds to the informal description of non-committing encryptions, presented in Section 3.1.2. A closer look, however, will show that the requirements from the simulator associated with a non-committing

encryption protocol (according to Definition 3.2) 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.

3.3 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 3.3.1 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 3.3.2 we present our construction of non-committing encryption. We use the following result, attributed to [BGW, CCD], as our starting point:[6]

Theorem 3.3 *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.*

3.3.1 Adaptively secure computation given non-committing encryption

Theorem 3.4 *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): 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

[6] A proof of this result can be extracted from Chapter 4, which deals with the more involved asynchronous model.

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 2.12 and 3.2 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

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

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:

$$\text{IDEAL}_{f,\tilde{\mathcal{S}}\tilde{\mathcal{A}}}(\vec{x}) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}\mathcal{A}}(\vec{x}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi',\mathcal{A}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x}) \tag{3.1}$$

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. (3.1).

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. (3.1). □

3.3.2 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. Such an initial set-up 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 uses a collection of trapdoor permutations together with a corresponding hard-core predicate [BM, Y2, GrL]. Actually, we need a collection of trapdoor permutation with the additional property that they are many permutations over the same domain.

[7] 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 3.2 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.

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 f' over D together with the trapdoor of f'. Such a collection is called a common-domain trapdoor system.

Definition 3.5 *A* common-domain trapdoor system *is an infinite set of finite permutations* $\{f_{\alpha,\beta}: D_\alpha \stackrel{1\text{-}1}{\mapsto} D_a\}_{(\alpha,\beta)\in P}$, *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 algorithm* 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 algorithm* 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.

Remarks:

- 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.
- Common-domain trapdoor systems can be constructed based on an arbitrary family of trapdoor permutations, $\{f_\beta: D_\beta \stackrel{1\text{-}1}{\mapsto} D_\beta\}$, 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., $|D_\beta| \geq \frac{1}{\mathrm{poly}(|\beta|)} \cdot 2^{|\beta|}$). We construct a common-domain family where the domain is $\{0,1\}^n$ and the permutations are natural extensions 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 amplification techniques (e.g., [Y2, GILVZ]) we obtain a proper common-domain system.

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. The scheme consists of two stages. In the first stage, called the key generation stage, the parties arrive at a situation where the sender has two trapdoor permutations f_a, f_b of a common-domain system, the trapdoor of only *one* of which is known to the receiver. Furthermore, the simulator will be able to generate, in a simulated execution of the protocol, two trapdoor permutations with the same distribution as in a real execution and such that the trapdoors of both permutations are known. (The simulator will later open dummy ciphertexts as either '0' or '1' by claiming that the decryption 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 by the simulator and never revealed). The key generation stage is independent of the bit to be transmitted (and can be performed before this bit is even determined).

Our most general implementation of this stage, based on any common-domain system, requires participation of all parties. It is described in Section 3.3.2. In the implementations based on the RSA and DH assumptions (see Section 3.3.3) the key-generation stage consists of only one message sent from the receiver to the sender.

The second stage, in which the actual transmission takes place, consists of only one message sent from the sender to the receiver. This stage consists of encryption and decryption algorithms, invoked by the sender and the receiver respectively.

We first present, in Section 3.3.2, the encryption and decryption algorithms as well as observations that will be instrumental for the simulation. In Section 3.3.2 we present the key generation protocol. (A reader that is satisfied with a construction based on specific number theoretic assumptions may, for simplicity, skip Section 3.3.2 and read Section 3.3.3 instead.) Finally we show that these together constitute the desired non-committing encryption protocol.

Encryption and decryption

Let f_a and f_b be two randomly selected permutations over the domain D, and let B be 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, the sender applies 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 3.6 *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,*

$$|\text{Prob}(B(f_b^{-1}(f_a(x))) = B(x)) - \frac{1}{2}| \quad \textit{is negligible} \tag{3.2}$$

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

Proof: 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.) □

From this point on, we assume that the pair (f_a, f_b) satisfies Eq. (3.2).

Lemma 3.7 *Let $\vec{y} = y_1, \ldots, y_{16k}$ be a random encryption of a bit σ. Then with probability $1 - 2^{-\Omega(k)}$ the bit decrypted from $\vec{y}$ is σ.*

Proof: 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. (3.2) 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 $2^{-\Omega(k)}$. □

Simulation assuming knowledge of both trapdoors. In Lemma 3.9 (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 at random as specified in the encryption algorithm.
- 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.
- 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).

Let us first prove a simple technical claim, that will help us in proving Lemma 3.9. Let BIN_m denote the binomial distribution over $[m]$.

Claim 3.8 *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 $\text{Prob}(\mu = i) = \text{BIN}_{8k}(i + 2k)$ for $i = 1, ..., 4n$ and $\text{Prob}(\mu = 0)$ 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 $2^{-\Omega(k)}$. □

Lemma 3.9 *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. $(0; \vec{x}^{(0)}, \phi^{(0)}; \vec{z}; r^{(0)}, f_{r^{(0)}}^{-1}) \stackrel{c}{\approx} \lambda_0$.
2. $(1; \vec{x}^{(1)}, \phi^{(1)}; \vec{z}; r^{(1)}, f_{r^{(1)}}^{-1}) \stackrel{c}{\approx} \lambda_1$.

Here $\stackrel{c}{\approx}$ stands for 'computationally indistinguishable'. We stress that the *same* dummy ciphertext, $\vec{z}$, appears in both (1) and (2).

Proof: Before describing how the dummy ciphertext $\vec{z}$ and the rest of the data are constructed, we summarize, in Figure 3-1, 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 $\tilde{\text{BIN}}_{8k}$ denotes the distribution of the number of '1's in $B(f_b^{-1}(y_i))$ for

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

Figure 3-1: 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.)

$i = 1, ..., 8k$. Eq. (3.2) implies that the statistical difference between BIN_{8k} and $\tilde{\text{BIN}}_{8k}$ 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 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[8]

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 $\gamma_{\psi^{(\sigma)}(i)}^{(0)} = 1 - \sigma$.)

[8] 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.

3. $\gamma^{(\rho^{-1}(b))}_{\psi^{(\sigma)}(i)} = B(f_b^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_b^{-1}(z_{\phi^{(\sigma)}(i)})) = B(x_i^{(\sigma)}) = \sigma$, for $i = 8k+1, ..., 13k$.
(E.g., if $\rho(0) = a$ this means $\gamma^{(1)}_{\psi^{(\sigma)}(i)} = \sigma$.)

4. $\gamma^{(\rho^{-1}(b))}_{\psi^{(\sigma)}(i)} = B(f_b^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_b^{-1}(z_{\phi^{(\sigma)}(i)})) = B(x_i^{(\sigma)}) = 1-\sigma$, for $i = 13k+1, ..., 16k$.
(E.g., if $\rho(0) = a$ this means $\gamma^{(1)}_{\psi^{(\sigma)}(i)} = 1 - \sigma$.)

5. Let $I = [8k]$ if $\rho(\sigma) = b$ and $I = \{8k+1, ..., 16k\}$ otherwise. Then, $\gamma^{(\sigma)}_{\psi^{(\sigma)}(i)} = B(f^{-1}_{\rho(\sigma)}(v_{\psi^{(\sigma)}(i)})) = B(f^{-1}_{\rho(\sigma)}(z_{\phi^{(\sigma)}(i)})) = B(f^{-1}_{\rho(\sigma)}(f_{\rho(1-\sigma)}(x_i^{(\sigma)})))$ 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 $\text{Prob}(\gamma^{(0)}_{\psi^{(\sigma)}(i)} = 1) \approx \frac{1}{2}$ for $i = 8k+1, ..., 16k$, whereas for $\rho(0) = a$ and $\sigma = 1$ we have $\text{Prob}(\gamma^{(1)}_{\psi^{(\sigma)}(i)} = 1) \approx \frac{1}{2}$ for $i = 1, ..., 8k$.)

This allows setting $\phi^{(\sigma)} = \psi \circ (\psi^{(\sigma)})^{-1}$ so that $x^{(\sigma)}_{\phi^{(\sigma)}(i)}$ 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

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

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

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 3-2. Suppose $\rho(0) = a$. Then, we may set

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

Figure 3-2: The distribution of the $\gamma^{(0)}$'s and $\gamma^{(1)}$'s. (μ is as in Claim 3.8.)

$\psi^{(0)}([8k]) = I_1 \cup I_2$ 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 3-3). 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 $S_4^0 = \sum_{i=8k+1}^{16k} \gamma_i^{(0)}$ we observe that it is distributed as $2k + \mu = \tilde{\mu}$ (of Claim 3.8) which in turn is statistically close to $\tilde{\text{BIN}}_{8k}$. We

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

Figure 3-3: 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.

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 $\pi : I_1 \cup I_2 \stackrel{1\text{-}1}{\mapsto} I_1 \cup I_3$, 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 3-4). In case

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

Figure 3-4: 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.

$\rho(0) = b$ we set $\psi^{(0)}([8k]) = I_3 \cup I_4$, $\psi^{(0)}(\{8k+1, ..., 16k\}) = I_1 \cup I_2$, $\psi^{(1)}([8k]) = I_2 \cup I_4$ 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 $\sum_{i=8k+1}^{16k} \gamma_{\psi^{(\sigma)}(i)}^{(\sigma)} \stackrel{\mathrm{d}}{=} \tilde{\mathrm{BIN}}_{8k}$. This is indeed consistent with the requirement that these $\gamma_{\psi^{(\sigma)}(i)}^{(\sigma)}$'s are almost uniformly and independently distributed. But this is not sufficient. In particular, we also need $\sum_{i \in J} \gamma_{\psi^{(\sigma)}(i)}^{(\sigma)} \stackrel{\mathrm{d}}{=} \tilde{\mathrm{BIN}}_{3k}$, where $J = \{8k < i \leq 16k : \gamma_{\psi^{(\sigma)}(i)}^{(1-\sigma)} = 1 - \sigma\}$ and furthermore the above sum needs to be independent of $\sum_{i \in \{8k+1, ..., 16k\} - J} \gamma_{\psi^{(\sigma)}(i)}^{(\sigma)}$ (which in turn should be statistically close to BIN_{5k}). Let us start with the case $\sigma = 0$. In this case we need

$$\sum_{i \in J} \gamma_i^{(0)} \stackrel{\mathrm{d}}{=} \tilde{\mathrm{BIN}}_{3k}, \tag{3.3}$$

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 3-2 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. (3.3) we construct a random variable $\mu' \in \{0, 1, ..., k\}$ (analogously to Claim 3.8) so that $p_j \stackrel{\text{def}}{=} \text{Prob}(\mu' = j) = \text{BIN}_{3k}(k + j)$ for $j \in [k]$ (with the rest of the mass on $\mu' = 0$) and constrain the $\gamma_i^{(0)}$'s to satisfy $\text{Prob}(\sum_{i \in J'} \gamma_i^{(0)} = j) = p_j$. We get $\sum_{i \in J} \gamma_i^{(0)} = k + \mu' \stackrel{\text{d}}{=} \tilde{\text{BIN}}_{3k}$ (analogously to Claim 3.8). 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 $\mu'' \stackrel{\text{d}}{=} \text{BIN}_{5k} - 2k$ 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 $\text{Prob}(\mu < \mu') < \text{Prob}(\mu < k) = 2^{-\Omega(k)}$).

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

$$\sum_{i \in J} \gamma_i^{(1)} \stackrel{\text{d}}{=} \tilde{\text{BIN}}_{3k}, \tag{3.4}$$

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. (3.4) we restrict the $\gamma_i^{(1)}$'s in $J' \stackrel{\text{def}}{=} J \cap I_1$ 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

$$(\sigma; \vec{x}^{(\sigma)}, \phi^{(\sigma)}; \vec{z}; r^{(\sigma)}, f_{r^{(\sigma)}}^{-1}) \stackrel{\text{c}}{\approx} \lambda_\sigma = (\sigma; \vec{x}, \phi; \vec{y}; r, f_r^{-1}). \tag{3.5}$$

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 $\mathrm{Prob}(B(x) = 0) \approx \frac{1}{2}$ (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). □

Key generation

We describe how the keys are generated, based on any common-domain trapdoor system. We use Oblivious Transfer [MRa1, EGL] in our constructions. Oblivious Transfer (OT) is a protocol executed by a sender S with inputs s_1 and s_2, and by a receiver R with input $\tau \in \{1, 2\}$. After executing an OT protocol, 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 learns s_1 or s_2. We are only concerned with the case where R is uncorrupted and non-erasing.

We use the implementation of OT described in [GMW] (which in turn originates in [EGL]). This implementation has an additional property, discussed below, that is useful in our construction. For self containment we sketch, in Figure 3-5, the [GMW] protocol for OT of one bit.

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 $\frac{1}{2}$. [9] 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.

The 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

[9] 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.

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 $f : D \to D$ 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 3-5: The [GMW] Oblivious Transfer protocol

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.

In Figure 3-6 we describe our key generation protocol. This protocol is valid as long as at least one party remains uncorrupted.

Simulation (Adaptive security of the encryption protocol)

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

Theorem 3.10 *Protocol ε is an $(n-1)$-resilient non-committing encryption protocol for n parties, in the presence of non-erasing parties.*

Proof (sketch): 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 $\text{IDEAL}_{\text{BTR}_{s,r},\mathcal{S}^{\mathcal{A}}}(\sigma) \stackrel{\text{d}}{=} \text{EXEC}_{\varepsilon',\mathcal{A}}(\sigma)$, 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

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 3-6: The key generation protocol

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 3.9, 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 3.9 and from the properties of the [GMW] Oblivious Transfer protocol. □

3.3.3 Alternative implementations of non-committing encryption

We describe two alternative implementations of our non-committing encryption scheme, based on the RSA and DH assumptions, respectively. These implementations have the advantage that the key generation stage can be simplified to consist of a single message sent from the receiver to the sender.

An implementation based on RSA. We first construct the following common-domain trapdoor system. The common domain, given security parameter n, is $\{0,1\}^n$. A permutation over $\{0,1\}^n$ is chosen as follows. First choose a number N uniformly from $[2^{n-1}...2^n]$, *together with its factorization* (via Bach's algorithm [Ba]). Next choose a prime $2^n < e < 2^{n+1}$. (This way, we are assured that $gcd(e, \phi(N)) = 1$, where $\phi()$ is Euler's totient function, even if the factorization of N is not known.) Let $f_N(x) = x^e (\mathrm{mod} N)$ if $x < N$ and $f_N(x) = x$ otherwise. With non-negligible probability N is a product of two large primes. Thus, this construction yields a collection of common-domain permutations which are weakly one-way. Employing an amplification procedure (e.g., [Y2, GILVZ]) we obtain a proper common-domain system.

This common-domain trapdoor system can be used as described in Section 3.3.2. However, here the key-generation stage can be simplified considerably. Observe that it is possible to choose a permutation from the above distribution *without knowing its trapdoor.* Specifically, this is done by choosing the numbers N of the different instances of f_N in the direct way, without knowing their factorization. Thus, the receiver will choose two trapdoor permutations f_a, f_b, where only the trapdoor to f_r (i.e., f_r^{-1}) is known, $r \in_R \{a,b\}$. Both f_a, f_b are now sent to the sender, who proceeds as in Section 3.3.2. In a simulated execution the simulator will choose both f_a and f_b together with their trapdoors.[10]

An implementation based on DH. Consider the following construction. Although it fails to satisfy Definition 3.5, it will be 'just as good' for our needs. The common domain, given security parameter n, is a prime $2^{n-1} < p < 2^n$ where the factorization of $p-1$ is known. Also, a generator g of Z_p^* is fixed. p and g are publicly known. All computations are done modulo p. To choose a permutation over Z_p^*, choose an element $v \in_R \mathcal{Z}_{p-1}^*$ and let $f_v(x) = x^v$. The public description of f_v is $y \triangleq g^v$. The 'trapdoor' is $u \triangleq v^{-1} (\mathrm{mod} p-1)$.

[10] A similar idea was used in [DP].

This construction has the following properties:

- Although it is hard to compute f_v if only p, g, y are known, it is easy to generate random elements $x \in_R Z_p^*$ together with $f_v(x)$: choose $z \in_R Z_p^*$, and set $x = g^z$ and $f_v(x) = y^z$. (This holds since $f_v(x) = x^v = g^{zv} = y^z$.)
- If u is known then it is easy to compute $f_v^{-1}(x) = x^u$.
- An algorithm A that inverts f_v given only p, g, y can be easily transformed into an algorithm A' that given p, g, g^α, g^β outputs $g^{\alpha\beta}$ (that is, into an algorithm that contradicts the Diffie-Hellman (DH) assumption). Specifically, Assume that $A(p, g, g^v, x^v) = x$. Then, on input p, g, g^α, g^β, algorithm A' will run $A(p, g^\alpha, g, g^\beta)$ to obtain $g^{\alpha\beta}$.
- It is possible to choose a permutation from the above distribution *without knowing its trapdoor.* Specifically, this is done by uniformly choosing numbers $y \in_R Z_p^*$ until a generator is found. (It is easy to decide whether a given y is a generator of Z_p^* when the factorization of $p-1$ is known.)

Note that both in the encryption process and in the simulation it is not necessary to compute the permutations f on arbitrary inputs. It suffices to be able to generate random elements x in the domain together with their function value $f(x)$. Thus, this construction is used in a similar way to the previous one.

A concluding remark to Section 3.3. Our solutions for non-erasing parties may appear somewhat unsatisfactory since they are based on 'trusting' the receiver to choose trapdoor permutations without knowing the trapdoor, whereas the permutation can be chosen together with its trapdoor by simple 'honest-looking' behavior. Recall, however, that if honest-looking parties are allowed then no (non-trivial) protocol can be proven adaptively secure (via black-box simulation if claw-free pairs exist). We do not see a meaningful way to distinguish between the 'honest-looking behavior' that foils the security of our constructions and the 'honest-looking behavior', described in Section 3.1.1, that foils provability of the adaptive security of any protocol.

3.4 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 3.3 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.)