Somewhat Non-Committing Encryption and Efficient Adaptively Secure Oblivious Transfer

Juan A. Garay* Daniel Wichs† Hong-Sheng Zhou‡

December 19, 2008

Abstract

Designing efficient cryptographic protocols tolerating adaptive adversaries, who are able to corrupt parties on the fly as the computation proceeds, has been an elusive task. Indeed, thus far no *efficient* protocols achieve adaptive security for general multi-party computation, or even for many specific two-party tasks such as oblivious transfer (OT). In fact, it is difficult and expensive to achieve adaptive security even for the task of *secure communication*, which is arguably the most basic task in cryptography.

In this paper we make progress in this area. First, we introduce a new notion called *second-corruption adaptive* security which is slightly stronger than static security but *significantly weaker than fully adaptive security*. As its names indicates, it limits the adversary's dynamic behavior to only one corruption – the second, in the case of two-party tasks – and as such, it is much easier to achieve. We then give a simple, generic protocol compiler which transforms any second-corruption secure protocol into a fully adaptively secure one. The compilation effectively decomposes the problem of adaptive security into two (simpler) problems which can be tackled separately: the problem of second-corruption adaptive security and the problem of realizing a weaker variant of secure channels.

We solve the latter problem by means of a new primitive that we call *somewhat non-committing encryption* resulting significant efficiency improvements over the standard method for realizing (fully) secure channels using (fully) non-committing encryption. Somewhat non-committing encryption has two parameters: an equivocality parameter ℓ (measuring the number of ways that a ciphertext can be "opened") and the message sizes k. Our implementation is very efficient for small values ℓ, *even* when k is large. This translates into a very efficient compilation of many second-corruption adaptively secure protocols (in particular, for a task with small input/output domains such as bit-OT) into a fully adaptively secure protocol.

Finally, we showcase the power of our methodology by applying it to the recent Oblivious Transfer protocol by Peikert *et al.* [Crypto 2008], which is only secure against static corruptions, to obtain the first efficient (expected-constant round, expected-constant number of public-key operations) and adaptively secure bit-OT protocol.

*AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932, USA. Email: garay@research.att.com.

†Computer Science Department, New York University, NY 10012, USA. Email: wichs@cs.nyu.edu.

‡University of Connecticut, Computer Science & Engineering, Storrs, CT 06269, USA. Email: hszhou@cse.uconn.edu.

1 Introduction

When defining the security of cryptographic protocols, we generally strive to capture as wide a variety of adversarial attacks as possible. The most popular method of doing so is the *simulation paradigm* [GMW87] where the security of a real-world protocol is compared to that of an ideal-world (perfectly secure) implementation of the same task. Within the simulation paradigm there are several flavors. Firstly, basic simulation only guarantees security for single copy of a protocol executing in isolation. The *Universal Composability* (UC) framework [Can01, Can05] extends the simulation paradigm and defines security for protocols executed in arbitrary environments, where executions may be concurrent and even maliciously interleaved. Secondly, we generally distinguish between *static* and *adaptive* security. Static security protects against an adversary who controls some fixed set of corrupted parties throughout the computation. Adaptive security, on the other hand, defends against an adversary who can corrupt parties adaptively at any point during the course of the protocol execution (for example by bribing them or hacking into their machines). For adaptive security, we also make a distinction between the *erasure model*, where honest parties are trusted to securely erase data as mandated by the protocol, and the *non-erasure model*, where no such assumptions are made. Given the difficulty of erasing data securely it is valuable to construct protocols in the latter model, which is the subject of this work.

The seminal result of [CLOS02] shows that it is theoretically possible to design an adaptively secure and universally composable protocol for almost any task assuming the presence of some trusted setup such as a randomly selected common reference string (CRS). Unfortunately, the final protocol of [CLOS02] should be viewed as a *purely theoretical* construction. Its reliance on expensive Cook-Levin reductions precludes a practical implementation. Alternative efficient approaches to two-party and multi-party computation received a lot of attention in the recent works of [DN03, GMY04, JS07, LP07, IPS08]. However, all of these results sacrifice some aspect of security to get efficiency. Concretely, the work of [LP07] only provides stand-alone static security, [JS07] provides UC static security, [GMY04] provides UC adaptive security but only in the erasure model, and [DN03] provides UC adaptive security but only for an honest majority. The recent work of [IPS08] *can* provide UC adaptive security but only given an efficient adaptively secure Oblivious Transfer (OT) protocol. However, as we will discuss, no such protocols were known.

Indeed, thus far *no* efficient protocols for general multi-party computation, or even for many specific two-party function evaluation tasks, achieve adaptive security. This is not surprising given the difficulty of realizing adaptive security for even the most fundamental task in cryptography: *secure communication*. As was observed in [CFGN96], standard security notions for encryption do not suffice. Adaptively secure communication schemes, also called *non-committing encryption* schemes, were introduced and constructed in [CFGN96] and studied further in [Bea97, DN00], but these protocols are fairly complicated and inefficient for large messages.

It turns out that many useful two-party tasks (e.g., Oblivious Transfer, OR, XOR, AND, Millionaires' problem, etc.) are *strictly harder* to achieve than secure communication, the reason being that these tasks allow two honest parties to communicate by using the corresponding ideal functionality. For example, using Oblivious Transfer (OT), an honest sender can transfer a message to a receiver by setting it as *both* of his input values. Therefore, an adaptively secure OT protocol for the transfer of k bit messages can be used as a non-committing encryption of a k bit message and so all of the difficulty and inefficiency of non-committing encryption must **also** appear in protocols for tasks such as OT. Further, unlike secure communication, many tasks *also* require security against the active and malicious behavior of the *participants*. This might lead us to believe that the two difficulties will be compounded making efficient adaptively secure implementations of such tasks infeasible or too complicated to contemplate.

Taking Oblivious Transfer as an example, this has indeed been the case in the past. We are aware of only two examples (albeit inefficient) of adaptively secure OT protocols, from [Bea98] and [CLOS02]. Both of these works first construct an OT protocol for the honest-but-curious setting and then compile it into a fully-secure protocol using generic and inefficient zero knowledge proofs. In both constructions, the underlying honest-but-curious OT protocols rely on ideas from non-committing encryption[1] and hence inherit its complexity. Since the

[1]The protocol of [Bea98] implicitly uses the plug-and-play approach from [Bea97], while the protocol of [CLOS02] uses non-committing encryption in a generic way.

full constructions require us to run zero knowledge proofs *on top of* the complex underlying honest-but-curious protocol, there is little hope of making them efficient by only using proofs for simple relations. This is in contrast to static security (and adaptive security in the erasure model) for which we *have* recently seen efficient constructions of OT protocols. For example, [GMY04, JS07, DNO08] construct OT protocols by only using simple and efficient zero-knowledge proofs, while two very recent and efficient protocols [PVW08, Lin08] do not use zero-knowledge proofs at all! The protocol of [PVW08] is particularly exciting since it is a UC-secure protocol in the CRS model which runs in two rounds and uses a constant number of public key operations. Achieving adaptive security based on these protocols has, however, remained as an open problem.

In summary, we can use ideas from non-committing encryption to get honest-but-curious adaptively secure OT protocols, *or* we can use various clever ideas to achieve static security in the malicious setting, but there is no known way to *combine* these techniques!

Our contributions. In this work we construct the *first* efficient (constant round, constant number of public-key operations) adaptively secure Oblivious Transfer protocol in the non-erasure model. Along the way we develop several techniques of independent interest which are applicable to adaptive security in general.

First, we introduce a new notion called *second-corruption adaptive* security which is slightly stronger than static security but significantly weaker than fully adaptive security. In particular, a second-corruption adaptively secure protocol for a task like OT, *does not* yield a non-committing encryption scheme and hence does not (necessarily) inherit its difficulty. We then give a generic compiler which transforms any second-corruption adaptively secure protocol into a (fully) adaptively secure protocol. The compiler is fairly simple: we take the original protocol and execute it over a secure communication channel (i.e., all communication from one party to another is sent over a secure channel). The compilation effectively decomposes the problem of adaptive security into two (simpler) problems which can be tackled separately: the problem of second-corruption adaptive security and the problem of realizing secure channels.

Unfortunately, we saw that the latter problem is a difficult one and existing solutions are not very efficient. Also, as we already mentioned, we cannot completely bypass this problem since adaptive security for many tasks *implies* secure channels. However, for the sake of efficiency, we would like to limit the use of secure channels (and hence the use of non-committing encryption) to a minimum. For example, we know that an OT protocol for one-bit messages implies a non-committing encryption of a one-bit message. However, to get adaptive security for a bit-OT protocol, our compiler, as described above, would use non-committing encryption to encrypt the *entire* protocol transcript, and hence much more than one bit!

We fix this discrepancy by introducing a new notion called *somewhat non-committing encryption*. Somewhat non-committing encryption has two parameters: the equivocality ℓ (measuring just how *non-committing* the scheme is) and the message size k. We first observe that somewhat non-committing encryption is efficient for small values of the equivocality parameter ℓ, *even* when k is large (i.e., when we encrypt long messages). Secondly, we observe that our compiler can use somewhat non-committing encryption where the equivocality ℓ is proportional to the size of the input and output domains of the functionality. As a result, we obtain a very efficient compiler transforming any second-corruption adaptively secure protocol for a task with small input/output domains (such as bit-OT) into a fully adaptively secure protocol.

We apply our methodology to the OT protocol of Peikert *et al.* [PVW08], resulting in the first efficient and adaptively secure bit-OT protocol. Peikert *et al.* actually present a general framework for constructing static OT, and instantiate this framework using the Quadratic Residuocity (QR), Decisional Diffie-Hellman (DDH), and Lattice-based assumptions. In this work, we concentrate on the QR- and DDH-based schemes. We show that relatively small modifications suffice to make these schemes second-corruption adaptively secure. We then employ our compiler, using somewhat non-committing encryption, to convert them into (fully) adaptively secure OT protocols.

To avoid diluting the main ideas of the paper, we put all proofs as well as background material, efficiency considerations, and our enhanced version of the QR dual-mode cryptosystem in the appendices.

2 Somewhat Non-Committing Encryption and Adaptive Security

2.1 Adaptive security in two-party protocols

What are some of the challenges in achieving adaptive security for a two-party protocol? Let's assume that a protocol π between two parties P_0, P_1 realizes a task $\mathcal{F}$ with respect to static adversaries. That means that there is a static simulator which can simulate the three basic cases: both parties are honest throughout the protocol, exactly one party is corrupted throughout the protocol or both parties are corrupted throughout the protocol. To handle adaptive adversaries, we require two more capabilities from our simulator: the ability to simulate a *first corruption* (i.e., the case that both parties start out honest and then one of them *becomes* corrupted) and simulating the *second corruption* (i.e., the case that one party is already corrupted and the other party becomes corrupted as well).

Simulating the first corruption is often the harder of the two cases. The simulator must produce the internal state for the corrupted party in a manner that is consistent with the protocol transcript so far and with the actual inputs of that party (of which the simulator had no prior knowledge). Moreover, the simulator needs to have all the necessary trapdoors to continue the simulation *while* only one party is corrupted. Achieving both of these requirements at once is highly non-trivial and this is one of the reasons why efficient protocols for adaptively secure two-party computation have remained elusive.

Interestingly, the entire difficulty of simulating the first corruption goes away if the protocol π employs secure channels for all communication between parties. At a high level, the simulator does not have to do any work while both parties are honest, since the real-world adversary does not see any relevant information during this time! When the first party *becomes* corrupted, we can just run a static simulation for the scenario in which this party *was* corrupted from the beginning but acting honestly and using its input. Then, we can "lie" and pretend that this communication (generated *ex post facto*) actually *took place* over the secure channel when both parties were honest. The lying is performed by setting the internal state of the corrupted party accordingly. Since our lie corresponds to the simulation of a statically corrupted party (which happens to act honestly), all of the trapdoors are in place to handle future mischievous behavior by that (freshly corrupted) party. The only problem left is in handling the second corruption – but this is significantly easier! The simulator still has to come up with a consistent internal state for the second corrupted party, but *does not* need to put in any special trapdoors as we are not worried about two corrupted parties attacking each other. In other words, when a protocol employs idealized secure channels, simulating an adaptive adversary boils down to the task of simulating an adaptively determined second corruption, which is generally much easier. We define this weakened notion of adaptive security which we call *second-corruption adaptive* security formally:

Definition 2.1. *An adversary $\mathcal{A}$ attacking a two-party protocol π is called* second-corruption adaptive *if the adversary corrupts at least one of the parties prior to the start of the protocol execution (i.e., the first corruption is static). The other party can then be corrupted adaptively at any point in the execution (i.e., the second corruption is adaptive). We say that a two party protocol π for a task $\mathcal{F}$ is* second-corruption adaptively secure *if it UC-realizes $\mathcal{F}$ for any environment $\mathcal{Z}$ and any second-corruption adaptive adversary $\mathcal{A}$.*

Informally, our discussion above argued the following claim:

Claim 2.2. *(Simplified and incorrect version) Assume that a two-party protocol π for a task $\mathcal{F}$ is second-corruption adaptively secure. Then the protocol is also fully adaptively secure if all communication between the parties is sent over an idealized secure channel.*

Unfortunately, the above claim (even if it were true in its stated simplified form) does not yet help us much. The problem is that idealized secure channels are not a very realistic assumption and implementing such channels in the real world requires the inefficient use of *non-committing encryption* to encrypt the entire protocol transcript. Luckily, it turns out that we often do not need to employ fully non-committing encryption to make the transformation of Claim 2.2 hold. We define a weaker primitive called *somewhat non-committing encryption* and show that this primitive can be implemented with significantly greater efficiency than (fully) non-committing encryption. Finally, we show that somewhat non-committing encryption is often *good enough*

to transform a second-corruption adaptively secure protocol into a fully adaptively secure protocol when the sizes of the input/output domains are small.

2.2 Defining *somewhat* non-committing encryption

Let us first recall the notion of non-committing encryption from [CFGN96]. This is a protocol used to realize secure channels in the presence of an *adaptive* adversary. In particular, this means that a simulator can produce "fake" ciphertexts and later explain them as encryptions of *any possible* given message. Several non-committing encryption schemes have appeared in literature [CFGN96, Bea97, DN00] but the main disadvantage of such schemes is the computational cost. All of the schemes are interactive (which was shown to be necessary in [Nie02]) and the most efficient schemes require $\Omega(1)$ public-key operations (e.g. exponentiations) *per bit* of plaintext.[2]

We notice that it is often unnecessary to require that the simulator can explain a ciphertext as the encryption of *any* later-specified plaintext. Instead, we define a new primitive, which we call *somewhat non-committing* encryption, where the simulator is given a set of ℓ messages during the generation of the fake ciphertext and must later be able to plausibly explain the ciphertext as the encryption of *any one of those* ℓ *messages*. In a sense, we distinguish between two parameters: the plaintext size (in bits) k and the equivocality ℓ (the number of messages that the simulator can plausibly explain). For fully non-committing encryption, the equivocality and the message size are related by $\ell = 2^k$. Somewhat non-committing encryption, on the other hand, is useful in accommodating the case where the equivocality ℓ is very small, but the message size k is large.

Functionality $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$

The ideal functionality $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$ interacts with an initiator I and a responder R. The functionality is parameterized by a non-information oracle $\mathcal{N}$. The functionality verifies that $sid = (I, R, sid')$ is consistent for all received messages and ignores otherwise.

Channel setup: Upon receiving an input $(\texttt{ChSetup}, sid, I)$ from party I, initialize the machine $\mathcal{N}$ and record the tuple $(sid, \mathcal{N})$. Pass the message $(\texttt{ChSetup}, I)$ to R. In addition, pass this message to $\mathcal{N}$ and forward its output to $\mathcal{S}$.

Message transfer: Upon receiving an input $(\texttt{Send}, sid, P, m)$ from party P where $P \in \{I, R\}$, find a tuple $(sid, \mathcal{N})$ and, if none exists, ignore the message. Otherwise, send the message $(\texttt{Send}, sid, P, m)$ to the *other* party $\overline{P} = \{I, R\} - \{P\}$. In addition, invoke $\mathcal{N}$ with $(\texttt{Send}, sid, P, m)$ and forward its output to the adversary $\mathcal{S}$.

Corruption: Upon receiving a message $(\texttt{Corrupt}, sid, P)$ from the adversary, send $(\texttt{Corrupt}, sid, P)$ to $\mathcal{N}$ and forward its output to the adversary. After the first corruption, we stop the execution of $\mathcal{N}$ and give the adversary complete control over the functionality (i.e. the adversary $\mathcal{S}$ learns all inputs and can specify any outputs).

Figure 1: The parameterized secure-channel ideal functionality, $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$.

It is challenging to define an ideal-functionality for *somewhat* non-committing encryption, for the same reason that it is difficult to define ideal functionalities for many useful primitives like *witness indistinguishable* proofs of knowledge: the ideal world often captures a notion of security which is *too* strong. Here, we take the approach of [CK02] where ideal-world functionalities are weakened by the inclusion of a *non-information oracle* which is a PPT TM that captures the information leaked to the adversary in the ideal world. The ideal world functionality for secure channels in Figure 1, is parameterized using a non-information oracle $\mathcal{N}$ which gets the values of the exchanged messages m and outputs some side information to the adversary $\mathcal{S}$. The security of the secure channel functionality $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}}$ depends on the security properties required for the machine $\mathcal{N}$ and thus we can capture several meaningful notions. Let us first start with the most secure option which captures (fully) non-committing encryption.

[2] No such lower bound has appeared in literature and proving it, or providing a more efficient scheme, seems like an interesting though difficult open problem.

Definition 2.3. *Let $\mathcal{N}^{\text{full}}$ be the oracle, which, on input* $(\texttt{Send}, sid, P, m)$*, produces the output* $(\texttt{Send}, sid, P, |m|)$ *and, on any inputs corresponding to the* `ChSetup`, `Corrupt` *commands, produces no output. We call the functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}^{\text{full}}}$, or just $\mathcal{F}_{\text{SC}}$ for brevity, a* (fully) non-committing secure channel*. A real-world protocol which realizes $\mathcal{F}_{\text{SC}}$ is called a* non-committing encryption scheme (NCE).

In the above definition, the oracle $\mathcal{N}$ never reveals anything about messages m exchanged by two honest parties, even if (both of the) parties later get corrupted. Hence the functionality is fully *non-committing*. To define *somewhat* non-committing encryption we first start with the following definitions of non-information oracles.

Definition 2.4. *A machine $\mathcal{R}$ is called a* message-ignoring *oracle if, on any input* $(\texttt{Send}, sid, P, m)$*, it ignores the value m and processes only the input* $(\texttt{Send}, sid, P, |m|)$*. A machine $\mathcal{M}$ called a* message-processing *oracle if it has no such restrictions. We call a pair of machines $(\mathcal{M}, \mathcal{R})$* well-matched *if no PPT distinguisher $\mathcal{D}$ (with oracle access to either $\mathcal{M}$ or $\mathcal{R}$) can distinguish the message-processing oracle $\mathcal{M}$ from the message-ignoring oracle $\mathcal{R}$.*

We are now ready to define the non-information oracle used by a somewhat non-committing secure channel ideal functionality.

Definition 2.5. *Let $(\mathcal{M}, \mathcal{R})$ be a well-matched pair which consists of a message-processing and a message-ignoring oracle respectively. Let $\mathcal{N}^\ell$ be a (stateful) oracle with the following structure.*

- *Upon initialization, $\mathcal{N}^\ell$ chooses a uniformly random index $i \xleftarrow{\$} \{1, \ldots, \ell\}$. In addition it initializes a tuple of ℓ* independent *TMs: $\langle \mathcal{N}_1, \ldots, \mathcal{N}_\ell \rangle$ where $\mathcal{N}_i = \mathcal{M}$ and, for $j \neq i$, the machines $\mathcal{N}_j$ are independent copies of the message-ignoring oracle $\mathcal{R}$.*
- *Whenever $\mathcal{N}^\ell$ receives inputs of the form $(\texttt{ChSetup}, sid, P)$ or $(\texttt{Send}, sid, P, m)$, it passes the input to each machine $\mathcal{N}_i$ receiving an output y_i. It then outputs the vector $(y_1, \ldots, y_\ell)$.*
- *Upon receiving an input $(\texttt{Corrupt}, sid, P)$, the oracle reveals the internal state of the message-processing oracle $\mathcal{N}_i$ only.*

For any such oracle $\mathcal{N}^\ell$, we call the functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}^\ell}$ an ℓ-equivocal non-committing secure channel*. For brevity, we will also use the notation $\mathcal{F}_{\text{SC}}^{\ell}$ to denote $\mathcal{F}_{\text{SC}}^{\mathcal{N}^\ell}$ for some such oracle $\mathcal{N}^\ell$. Lastly, a real world protocol which realizes $\mathcal{F}_{\text{SC}}^{\ell}$ is called an* ℓ-equivocal non-committing encryption scheme (ℓ-NCE).

As before, no information about messages m is revealed during the "send" stage. However, the internal state of the message-processing oracle $\mathcal{N}_i$, which is revealed upon corruption, might be "committing". Nevertheless, a simulator can simulate the communication between two honest parties over a secure channel, as modeled by $\mathcal{F}_{\text{SC}}^{\ell}$, in a way that allows him to later explain this communication as any one of ℓ possibilities. In particular, the simulator creates ℓ message-processing oracles and, for every `Send` command, the simulator chooses ℓ distinct messages $m_1, \ldots, m_\ell$ that he passes to the oracles $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$ respectively. Since message-processing and message-ignoring oracles are indistinguishable, this looks indistinguishable from the side-information produced by $\mathcal{F}_{\text{SC}}^{\ell}$. Later, when a corruption occurs, the simulator can convincingly explain the entire transcript of communication to any one of the ℓ possible options, by providing the internal state of the appropriate message-processing oracle $\mathcal{M}_i$.

2.3 The ℓ-NCE construction

The construction of ℓ-NCE is based on a *simulatable public-key system*, which was defined in [DN00]. A simulatable public-key system is one in which it is possible to generate public keys obliviously, without knowing the corresponding secret key, and to explain an honestly (non-obliviously) generated public key as one which was obliviously generated. In a similar way, there should be a method for obliviously generating ciphertexts (without knowing any plaintext) and to explain honestly generated (non-oblivious) ciphertexts as obliviously generated ones. We review the syntax and security properties of such a scheme in Appendix B. Our ℓ-NCE

protocol construction, shown in Figure 2, uses a fully non-committing secure channel, but only to send a *very short* message during the setup phase. Hence, for long communications, our ℓ-NCE scheme is significantly more efficient than (full) NCE.

Let $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be an *oblivious public key system* and $\widetilde{\mathsf{KG}}$, $\widetilde{\mathsf{Enc}}$ be the corresponding *oblivious key generator* and *oblivious ciphertext generator* algorithms. Furthermore, let $(\mathsf{KG}^{\mathsf{sym}}, \mathsf{Enc}^{\mathsf{sym}}, \mathsf{Dec}^{\mathsf{sym}})$ be a symmetric key encryption scheme in which the ciphertexts are indistinguishable from uniformly random values of the same length.

Channel Setup. An initiator I sets up a channel with a receiver R as follows:

1. The initiator I sends a random index $i \in \{1, \ldots, \ell\}$ to R over a fully non-committing secure channel.
2. The initiator I generates ℓ public keys. For $j \in \{1, \ldots, \ell\} \setminus \{i\}$, the keys $pk_j \leftarrow \widetilde{\mathsf{KG}}()$ are sampled obliviously, while $(pk_i, sk_i) \leftarrow \mathsf{KG}()$ is sampled correctly. The keys $pk_1, \ldots, pk_\ell$ are sent to the responder R while I stores sk_i.
3. The responder R chooses a random key $K \leftarrow \mathsf{KG}^{\mathsf{sym}}$ and computes $C_i = \mathsf{Enc}_{pk_i}(K)$ correctly. In addition, R samples $C_j \leftarrow \widetilde{\mathsf{Enc}}_{pk_j}()$ obliviously for $j \in \{1, \ldots, \ell\} \setminus \{i\}$ and sends the ciphertexts $C_1, \ldots, C_\ell$ to I.
4. The initiator I decrypts the key $K \leftarrow \mathsf{Dec}_{sk_i}(C_i)$. Both parties store the tuple (K, i).

Encryption. An initiator I encrypts a message m to a receiver R as follows:

1. The initiator I computes $C_i \leftarrow \mathsf{Enc}^{\mathsf{sym}}_K(m)$ and chooses C_j for $j \in \{1, \ldots, \ell\} \setminus \{i\}$ as uniformly random and independent values of length $|C_i|$. The tuple $(C_1, \ldots, C_\ell)$ is sent to R.
2. The receiver R ignores all values other than C_i. It computes $m \leftarrow \mathsf{Dec}_K(C_i)$.

Figure 2: Construction of an ℓ-NCE protocol.

Theorem 2.6. *The protocol in Figure 2 is an ℓ-NCE scheme. Specifically it UC-realizes the $\mathcal{F}_{\mathrm{SC}}^{\ell}$ ideal functionality in the presence of an active and* adaptive *adversary.*

In Appendix C we analyze the efficiency of the above scheme with appropriate instantiations of the underlying secure-channel implementation (using full NCE), simulatable public-key system and symmetric key encryption scheme. We show that our scheme uses a total of (expected) $\mathcal{O}(\log \ell)$ public key operations, $\mathcal{O}(\ell)$ communication and (expected) constant rounds of interaction for the channel setup phase. After channel-setup, encryption is non-interactive and requires only symmetric-key operations. However, the encryption of a k bit message requires $\mathcal{O}(\ell k)$ bits of communication.

2.4 The adaptive security protocol compiler for SFE

As an application of ℓ-NCE, we give a general theorem, along the lines of Claim 2.2, showing that a protocol with second-corruption adaptive security can be compiled into a protocol with (full) adaptive security when all of the communication is encrypted using ℓ-NCE for some appropriate ℓ. However, we must first fix two problems which we ignored during the discussion leading to the statement of Claim 2.2.

Firstly, we know that most tasks cannot be realized in the Universal Composability framework without the use of *trusted setup*. However, the use of trusted setup complicates our transformation. The point of using (somewhat) non-committing encryption is that the simulator can lie about *anything that occurs while both parties are honest*. However, we often rely on trusted setup in which some information is given to the adversary even when both parties are honest. For example, the usual modeling of a common reference string (see Appendix A) specifies that this string is made public and given to the adversary even when *none* of the participants in the protocol are corrupted. In this case the simulator is committed to such setup even if the parties communicate over secure channels. Therefore our protocol compiler must start with a protocol where the simulation of trusted setup is independent of the corrupted party. Alternatively, we insist that the original

Functionality $\mathcal{F}_{\text{SFE}}$

The functionality $\mathcal{F}_{\text{SFE}}^{f}$ interacts with an *initiator* I and a *responder* R. The functionality verifies that all received messages share a consistent value $sid = (I, R, sid')$ and ignores them otherwise.

Input: Upon receiving the input value $(\texttt{Input}_I, sid, x_I)$ from the initiator I, record the value (I, x_I) and send the message $(\texttt{Input}_I, sid)$ to the adversary $\mathcal{S}$. Similarly, upon receiving the input value $(\texttt{Input}_R, sid, x_R)$ from the responser R, record the value (R, x_R) and send the message $(\texttt{Input}_R, sid)$ to the adversary $\mathcal{S}$.

Output: Upon receiving the message $(\texttt{Output}_I, sid)$ from the adversary $\mathcal{S}$, if either (I, x_I) or (R, x_R) is not recorded, ignore the message. Else compute $(y_I, y_R) = f(x_I, x_R)$ and send the output value $(\texttt{Output}_I, sid, y_I)$ to I. Similarly, upon receipt of $(\texttt{Output}_R, sid)$ send the output value $(\texttt{Output}_R, sid, y_R)$ to R.

Figure 3: A two-party secure evaluation of a function $f : X_I \times X_R \to Y_I \times Y_R$.

protocol allows for a first corruption which occur immediately *after* the setup is used (and hence the simulation of setup *must* be independent of the corrupted party).

The second problem is that even non-committing encryption *commits* the simulator to the lengths of the exchanged messages, the number of such messages, and the identifies of the sender and receiver of each message. Therefore we require that the initial protocol is *well-structured* so that this information is fixed and always the same. In other words, a protocol execution should have the same number of messages, message lengths and order of communication independent of the concrete inputs of the participating parties. Almost all known constructed protocols for cryptographic tasks are well-structured. Although this requirement is not very restrictive, it does require the introduction of some new terminology and we defer the formal definition to Appendix D. A well-structured protocol that uses some ideal-setup $\mathcal{T}$ must consist of two different phases: a *setup phase* and *communication phase*. All of the side-information that the adversary gets from $\mathcal{T}$ happens during the setup phase and *before* the parties communicate with each other.

Definition 2.7. *An adversary $\mathcal{A}$, who attacks a well-structured two-party protocol π using setup $\mathcal{T}$, is* second-corruption adaptive with respect to setup *if the adversary corrupts at least one of the parties* prior to *the beginning of the communication phase. The other party can then be corrupted adaptively at any point in the execution. We say that a well-structured two-party protocol π using setup $\mathcal{T}$ is* second-corruption adaptively secure with respect to setup *for a task $\mathcal{F}$, if it UC-realizes $\mathcal{F}$ for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$ which is* second-corruption adaptive with respect to setup*. For ease of exposition, we generally drop the term "*with respect to setup*" and make it the default notion for any protocol π which uses some setup $\mathcal{T}$.*

We now have all the definitions needed to formally state the correctness of our compilers for transforming second-corruption adaptively secure protocols into (fully) adaptively secure protocols. First we look at the simple compiler using idealized secure channels. We now state the corrected version of Claim 2.2.

Theorem 2.8. *Let $\mathcal{F}_{\text{SFE}}$ be the two-party ideal functionality which computes some function f as defined in Figure 3. Assume that a* well-structured *two-party protocol π for $\mathcal{F}_{\text{SFE}}$ is second-corruption adaptively secure. Let π' be the protocol in which the parties run π but only communicate with each other using non-committing secure channels as modeled by $\mathcal{F}_{\text{SC}}$. Then π' is (fully) adaptively secure.*

The intuition behind the above theorem was explained in Section 2.1. Also, as we mentioned, this compiler is usually not very efficient because of its excessive use of secure channels and hence NCE. Recall that secure channels are employed so that, when both parties are honest, the adversary does not see any useful information and so this case is easy to simulate. Then, when the first party gets corrupted, our simulator simply makes up the transcript of the communication that should have taken place *ex post facto*. This transcript is generated based on which party got corrupted, what its inputs were and what its outputs were. However, we notice that for many simple protocols there are not too many choices for this information. The simulator must simply be

able to credibly lie that the communication which took place over the secure channel corresponds to any one of these possible choices. Using this intuition, we are now ready to show that a more efficient compiler using ℓ-NCE (for some small ℓ) suffices.

Theorem 2.9. *Let $\mathcal{F}_{\text{SFE}}$ be the two-party ideal functionality computing some* deterministic *function $f : X_I \times X_R \to Y_I \times Y_R$, as defined in Figure 3. Assume that a well-structured two-party protocol π for $\mathcal{F}_{\text{SFE}}$ is second-corruption adaptively secure. Let π' be the protocol in which the parties run π but only communicate with each other using ℓ-equivocal secure channels as modeled by $\mathcal{F}_{\text{SC}}^{\ell}$ where $\ell = |X_I||Y_I| + |X_R||Y_R|$. Then π' is (fully) adaptively secure.*

Next we will apply our adaptive security compiler of Theorem 2.9 to the concrete problem of bit OT resulting in the first efficient protocol for this task.

3 Efficient and Adaptively Secure Oblivious Transfer

We start by giving an ideal functionality for OT. The one given in [PVW08] is adopted from [CLOS02], wherein the ideal-world adversary is in charge of delivery of input messages from the dummy parties to the ideal functionality. Here we choose to instead use the modeling of [Can05], and reformulate the OT functionality as presented in Figure 4. The multi-session version, $\hat{\mathcal{F}}_{\text{OT}}$, can be defined as in [CR03].

Functionality $\mathcal{F}_{\text{OT}}$

$\mathcal{F}_{\text{OT}}$ interacts with a sender S and a receiver R and the adversary $\mathcal{S}$. For each input or message, the functionality verifies that $sid = (R, S, sid')$; it ignores it otherwise.

- Upon receiving an input $(\texttt{Sender}, sid, x_0, x_1)$ from party S, where each $x_i \in \{0,1\}^\lambda$, record (x_0, x_1) and send $(\texttt{Sender}, sid)$ to the adversary. Ignore further $(\texttt{Sender}, \ldots)$ inputs.
- Upon receiving an input $(\texttt{Receiver}, sid, \sigma)$ from party R, where $\sigma \in \{0,1\}$, record σ and send $(\texttt{Receiver}, sid)$ to the adversary. Ignore further $(\texttt{Receiver}, \ldots)$ inputs.
- Upon receiving a message $(\texttt{Output}, sid)$ from the adversary, if either x_0, x_1 or σ is not recorded, ignore the message. Else send output $(\texttt{Output}, sid, x_\sigma)$ to R. Ignore further $(\texttt{Output}, \ldots)$ messages from the adversary.

Figure 4: The oblivious transfer ideal functionality, $\mathcal{F}_{\text{OT}}$.

3.1 The PVW oblivious transfer protocol

In [PVW08], Peikert *et al.* construct an efficient OT protocol in the CRS model with UC security against a malicious but static adversary. They do so by introducing a new primitive called a *dual-mode cryptosystem*, which almost immediately yields an OT protocol in the CRS model, and give constructions of this primitive under the DDH, QR and lattice hardness assumptions. We therefore first present a brief (informal and high-level) review of dual mode encryption as in [PVW08] and then will formally define a modified version of this primitive which will allow us to get adaptive security.

A dual-mode cryptosystem is initialized with *system parameters* which are generated by a trusted third party. For any choice of system parameters, the cryptosystem has two types of public/private key pairs: *left* key pairs and *right* key pairs. The key-generation algorithm can sample either type of key pair and the user specifies which type is desired. Similarly, the encryption algorithm can generate a *left* encryption or a *right* encryption of a message. When the key pair type matches the encryption type (i.e. a left encryption of a message under a left public key) then the decryption algorithm (which uses the matching secret key) correctly recovers the message.

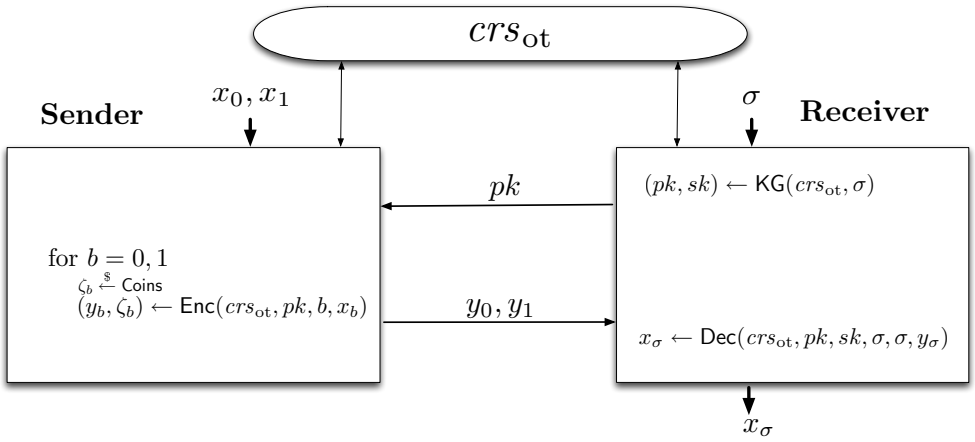


Figure 5: The generic OT protocol in [PVW08].

As shown in [PVW08], a dual-mode cryptosystem can be used to get an OT protocol as shown in Figure 5. The receiver chooses to generate a left or right key depending on his input bit σ, and the sender uses left-encryption ($b = 0$) for the left message x_0 and right-encryption for the right message. The receiver then uses the secret key to correctly decrypt the chosen message.

Security against malicious (static) adversaries in the UC model relies on the two different modes for generating the system parameters: *messy* mode and *decryption* mode. In *messy mode*, the system parameters are generated together with a *messy trapdoor*. Using this trapdoor, any public key (even one which is maliciously generated) can be easily labeled a left key or a right key. Moreover, in messy mode, when the encryption type does not match the key type (e.g. a left encryption using a right public key) then the ciphertext is statistically independent of the message. Messy mode is useful to guarantee security against a *corrupt receiver:* the messy trapdoor makes it easy to extract the receiver bit and to create a fake ciphertext for the message which should not be transferred. On the other hand, in *decryption mode*, the system parameters are generated together with a *decryption trapdoor* which can be used to decrypt both left and right ciphertexts. Moreover, in decryption mode, left public keys are statistically indistinguishable from right public keys. Decryption mode is useful to guarantee security against a *corrupt sender:* the decryption trapdoor is used to create a public key which completely hides the receiver's selection bit, and to compute a decryption trapdoor and extracting both of the sender's messages. In each mode, the security of one party (i.e., the sender in messy mode, and the receiver in decryption mode) is guaranteed information theoretically. To achieve security for both parties simultaneously all that is needed is one simple computational requirement: the system parameters generated in messy mode need to be computationally indistinguishable from those generated in decryption mode.

Next we consider the realization of multi-session functionality $\hat{\mathcal{F}}_{\mathrm{OT}}$. Based on the discussion above, each sender-receiver pair accesses to a CRS crs_{ot}. As shown in [PVW08], multiple bits can be obliviously transferred between the same pair of parties. However, for the case that multiple bits are obliviously transferred among arbitrary pairs of parties, we need a much longer CRS whose length is linear of the number of pairs. For example for n parties, the CRS will include $\binom{n}{2}$ independent copies of crs_{ot}. We note that the length of CRS could be reduced to n if we associate each sender (or each receiver), instead of each sender-receiver pair, a copy of crs_{ot}. A natural question comes: Can the CRS length be independent of n?

Without changing the two-move structure (which is one of the merits of PVW protocol), the answer is unclear. It seems that the simulator could use just two copies of crs_{ot}, one in the messy mode and the other in decryption mode, and based on its knowledge of the corrupted parties, for each sub-session involved by a corrupted sender (resp. receiver), the crs_{ot} in decryption (resp. messy) mode is supplied. However the environment can still notice the difference between the real and ideal worlds say by comparing the number of

corrupted senders and the number of the invoking one of the two $crs_{\rm ot}$ copies.

But if we are willing to give up the two-move structure, we can adopt the "stretching CRS" technique from [CR03]. Each sender-receiver pair can create $crs_{\rm ot}$ on the fly for the transferring stage by applying a coin tossing instead of querying the CRS functionality directly for $crs_{\rm ot}$; the coin tossing can be based on a UC commitment which can further be based on a constant CRS denoted as $crs_{\rm com}$, and good efficient candidate can be found in [DG03]; now the CRS for the multi-session protocol is $crs_{\rm com}$ whose length is constant. However the technique is only good for the case that $crs_{\rm ot}$ is uniformly distributed. In the case that $crs_{\rm ot}$ follows some non-uniform distribution, we separate $crs_{\rm ot}$ into two parts, a non-uniform part $crs_{\rm sys}$ and we call it *system CRS*[3], and a uniform part $crs_{\rm tmp}$ and we call it *temporal CRS*. If the $crs_{\rm ot}$ of all sub-sessions are only differentiated from the temporal parts and they share the non-uniform part, then we can still apply the previous method to obtain $crs_{\rm tmp}$ by applying a coin tossing based on a constant length $crs_{\rm com}$; now the CRS for the multi-session protocol is $(crs_{\rm sys}, crs_{\rm com})$ which is constant.

We note that in [PVW08] the DDH based protocol is under a uniform $crs_{\rm ot} = crs_{\rm tmp} = (g_0, h_0, g_1, h_1)$, where $crs_{\rm sys} = \emptyset$; for the QR-based protocol, $crs_{\rm ot} = (N, y)$ is not uniform, but it can be cut into a non-uniform part $crs_{\rm sys} = N$ which can be shared in the whole execution, and a uniform part $crs_{\rm tmp} = y$ which can generated on the fly by a sender-receiver pair. We can transform the both protocols into the ones realizing $\hat{\mathcal{F}}_{\rm OT}$ under constant length CRS and based on constant rounds [4] .

3.2 Second-corruption adaptively secure OT

In order to make the PVW OT protocol adaptively secure using our methodology, we need to make it second-corruption adaptively secure with respect to trusted setup (Section 2.4). We do so by a series of simple transformations.

First, we observe that in the PVW protocol, the simulator must choose the CRS based on which party is corrupt – i.e. the CRS should be in messy mode to simulate a corrupt receiver or in decryption mode to simulate a corrupt sender. As we mentioned earlier, second-corruption adaptive security requires that the simulator chooses the CRS in such a way that either party can later be corrupted.

Compared to the original PVW protocol, the transformed version discussed in the previous subsection has more advantage. Here $crs = (crs_{\rm sys}, crs_{\rm com})$ will be revealed to the environment. Note that $crs_{\rm sys}$ is independent of which party is corrupted since all $crs_{\rm ot}$ share the same system CRS. Further if the commitment protocol is adaptively secure, then $crs_{\rm com}$ is also independent of which party is corrupted. Hence the whole CRS can be simulated adaptively.

Second, we must consider the case where the sender starts out corrupted but the receiver might *also* become corrupted later on. In this case, to handle the corrupt sender, the simulator needs to simulate the execution in decryption mode. Moreover, to extract the sender's value, the simulator uses the decryption trapdoor to create a *dual public key* (on behalf of the receiver) which comes with both a left and a right secret key. Later, if the receiver becomes corrupted, the simulator needs to explain the randomness used by the receiver during key generation to create such a public key. Luckily, current dual-mode schemes already make this possible and we just update the definition with a property called *encryption key duality* to capture this.

Third, we consider the case where the receiver is corrupted at the beginning but the sender might *also* become corrupted later on. In this case the simulator simulates the execution in messy mode. In particular, the simulator uses the messy trapdoor to identify the receiver key type (right or left) and thus extract the receiver bit. Then the simulator learns the appropriate sender message for that bit and (honestly) produces the ciphertext for that message. In addition, the simulator must produce a "fake" ciphertext for the other message. Since, in messy mode, this other ciphertext is statistically independent of the message, it is easy to do so. However, if the sender gets corrupted later, the simulator must *explain* the fake ciphertext as an encryption of some particular message. To capture this ability, we require the existence of *internal state reconstruction* algorithm which can

[3] Now the former case that $crs_{\rm ot}$ is uniform can be viewed as a special case, i.e., $crs_{\rm ot} = (crs_{\rm sys}, crs_{\rm tmp})$ where $crs_{\rm sys}$ is empty.

[4] If we plug in the Damgard-Groth commitment, the round number is 6 for transferring 1-bit.

explain the fake ciphertext as an encryption of any message. Again, we notice that the QR instantiation of the PVW scheme already satisfies this new notion as well.

Enhanced Dual Mode Encryption. A dual-mode cryptosystem for message space $\{0,1\}^\lambda$ is defined by the following polynomial-time algorithms:

- $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mu)$. The parameter generation algorithm PG is a randomized algorithm which takes security parameter λ and mode $\mu \in \{\mathrm{mes}, \mathrm{dec}\}$ as input, and outputs (crs, τ), where crs is a common reference string and τ is the corresponding trapdoor information. For notational convenience, the random coins used for parameter generation are also included in τ. Note that our parameter generation PG includes two stages $\mathsf{PG}_{\mathrm{sys}}$ and $\mathsf{PG}_{\mathrm{tmp}}$, i.e., $(crs_{\mathrm{sys}}, \tau_{\mathrm{sys}}) \leftarrow \mathsf{PG}_{\mathrm{sys}}(1^\lambda, \mu)$ and $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$, $crs \leftarrow (crs_{\mathrm{sys}}, crs_t emp)$, and $\tau \leftarrow (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$.
- $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$. The key generation algorithm KG is a randomized algorithm which takes crs and a selection number $\sigma \in \{0,1\}$, and outputs a key pair (pk, sk), where pk is an encryption key and sk the corresponding decryption key. The random coins used for key generation are also included in sk.
- $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, b, m)$. The encryption algorithm Enc is a randomized algorithm which takes crs, pk, a branch number $b \in \{0,1\}$, and plaintext m, and outputs a ciphertext c. Here ζ is the random coins used for encryption.
- $m \leftarrow \mathsf{Dec}(crs, pk, sk, b, \sigma, c)$. The decryption algorithm Dec is a deterministic algorithm which takes crs, pk, sk, branch number b, and selection number σ, and in the case that $b = \sigma$, it outputs m.
- $\rho \leftarrow \mathsf{MessyId}(crs, \tau, pk)$. The messy branch identification algorithm $\mathsf{MessyId}$ is a deterministic algorithm which takes crs, τ and pk, and outputs the branch number ρ corresponding to the messy branch.
- $(c, \omega) \leftarrow \mathsf{FakeEnc}(crs, \tau, pk, b, \rho)$. The fake encryption algorithm $\mathsf{FakeEnc}$ is a randomized algorithm which takes crs, τ, pk, the branch number b, and the identified messy branch number ρ, and in the case that $b = \rho$, it outputs a fake ciphertext c and internal information ω.
- $\zeta \leftarrow \mathsf{Recons}(crs, \tau, pk, b, \rho, c, \omega, m)$. The internal state reconstruction algorithm Recons is a deterministic algorithm which takes $crs, \tau, pk, b, \rho, c, \omega$, and plaintext m, and outputs ζ.
- $(pk, sk_0, sk_1) \leftarrow \mathsf{DualKG}(crs, \tau)$. The dual key generation algorithm DualKG is a randomized algorithm, which takes crs, τ, and outputs an encryption keys pk, and two decryption key sk_0, sk_1 corresponding to branches 0 and 1, respectively.

Definition 3.1 (Enhanced Dual Mode Encryption)**.** *An enhanced dual mode cryptosystem is a tuple of algorithms as described above satisfying the following properties.*

COMPLETENESS: *For every $\mu \in \{0,1\}$, $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mu)$, $m \in \{0,1\}^\lambda$, $\sigma \in \{0,1\}$, and $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$, the decryption on branch σ is correct except with negligible probability; i.e., $\mathsf{Dec}(crs, pk, sk, \sigma, c) = m$, where $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, \sigma, m)$.*

ENHANCED MODE INDISTINGUISHABILITY: *The CRSes generated by PG in messy mode and in decryption mode are indistinguishable in the sense that (i) the both system CRSes are identically distributed, and (2) the two temporal CRSes are identical to some uniform distributions $\mathcal{U}_{\mathrm{mes}}$ and $\mathcal{U}_{\mathrm{dec}}$ respectively, and they are computationally indistinguishable, i.e.,*

$$\{crs_{\mathrm{sys}}\}_{(crs,\tau)\leftarrow\mathsf{PG}(1^\lambda,\mathrm{mes})} \overset{\mathrm{s}}{\approx} \{crs_{\mathrm{sys}}\}_{(crs,\tau)\leftarrow\mathsf{PG}(1^\lambda,\mathrm{dec})}$$

$$\{crs_{\mathrm{tmp}}\}_{crs_{\mathrm{tmp}}\overset{\$}{\leftarrow}\mathcal{U}_{\mathrm{mes}}(1^\lambda)} \overset{\mathrm{s}}{\approx} \{crs_{\mathrm{tmp}}\}_{(crs,\tau)\leftarrow\mathsf{PG}(1^\lambda,\mathrm{mes})} \overset{\mathrm{c}}{\approx} \{crs_{\mathrm{tmp}}\}_{(crs,\tau)\leftarrow\mathsf{PG}(1^\lambda,\mathrm{dec})} \overset{\mathrm{s}}{\approx} \{crs_{\mathrm{tmp}}\}_{crs_{\mathrm{tmp}}\overset{\$}{\leftarrow}\mathcal{U}_{\mathrm{dec}}(1^\lambda)}$$

MESSY BRANCH IDENTIFICATION AND CIPHERTEXT EQUIVOCATION: *For every $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mathrm{mes})$ and every pk, $\mathsf{MessyId}(crs, \tau, pk)$ outputs a branch value ρ such that for every $m \in \{0,1\}^\lambda$, $\mathsf{Enc}(crs, pk, \rho, \cdot)$ is simulatable, i.e.,*

$$\{c, \zeta\}_{(c,\zeta)\leftarrow\mathsf{Enc}(crs,pk,\rho,m)} \overset{\mathrm{s}}{\approx} \{c, \zeta\}_{(c,\omega)\leftarrow\mathsf{FakeEnc}(crs,\tau,pk,\rho),\zeta\leftarrow\mathsf{Recons}(crs,\tau,pk,\rho,c,\omega,m)}$$

ENCRYPTION KEY DUALITY: *For every* $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mathrm{dec})$*, there exists* $(pk, sk_0, sk_1) \leftarrow \mathsf{DualKG}(crs, \tau)$ *such that for every* $\sigma \in \{0,1\}$*,* (pk, sk_σ) *is statistically indistinguishable from the honestly generated key pair, i.e.,*

$$\{pk, sk_\sigma\}_{(pk,sk_0,sk_1)\leftarrow\mathsf{DualKG}(crs,\tau)} \overset{\mathrm{s}}{\approx} \{pk, sk\}_{(pk,sk)\leftarrow\mathsf{KG}(crs,\sigma)}$$

Construction. Based on the above transformations, a generic construction for a second-corruption adaptively secure OT protocol is given in Figure 6. It consists of two phases, the coin tossing phase and the transferring phase (which is separated by a dot line in the figure). The CRS consists of two pieces; the first piece is a system CRS denoted as crs_{sys}; and the second piece is for an adaptively secure UC commitment protocol[5] which will be used for constructing a coin tossing protocol. The UC commitment includes two stages, the commit and the open stages which could be interactive; a randomly selected value r is committed by the receiver for the sender in the commit stage, and after receiving a randomly selected value s from the sender, the receiver open the committed r to the sender, and both sender and the receiver can compute a temporal CRS t based on s and r. The temporal CRS t together with the system CRS crs_{sys} will be used as the CRS for the transferring phase and we denote it as crs_{ot}. With crs_{ot} in hand, we plug in the PVW protocol in Figure 5 but based on the enhanced dual-mode cryptosystem to achieve message transferring.

Theorem 3.2. *Given an adaptively UC-secure commitment scheme and an enhanced dual-mode cryptosystem as in Definition 3.1, the multi-session protocol based on Figure 6 is second-corruption adaptively secure to realize* $\hat{\mathcal{F}}_{\mathrm{OT}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid model.*

By "plugging in" efficient instantiations of the two building blocks above, we obtain efficient concrete protocols for second-corruption adaptively secure OT. For example, good candidates for adaptively secure UC commitments can be found in [DG03], while a QR-based dual-mode encryption scheme is presented in [PVW08]. In Section Appendix F, we show that this scheme also satisfies Definition 3.1. We note that a second-corruption adaptively secure OT protocol based on the DDH assumption. In this case, however, in order to make ciphertext equivocation possible, we also need an efficient Σ-protocol for the equality of discrete logs.

3.3 Final adaptively secure OT protocol

We now apply our compiler from Section 2.4 to the protocol in Figure 6, to immediately obtain an efficient an adaptively secure OT protocol in the UC framework. Namely, we get a protocol which runs in (expected) constant number of rounds and uses (expected) constant number of public key operations.

Acknowledgements

We thank Ran Canetti, Yevgeniy Dodis, Stas Jarecki and Aggelos Kiayias for their help. Specially, an early discussion with Ran advances this work.

References

[Bea97] Donald Beaver. Plug and play encryption. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 1997.

[Bea98] Donald Beaver. Adaptively secure oblivious transfer. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 1998.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[5] In fact, a second-corruption adaptively secure UC commitment would suffice here since the CRS used is insensitive.

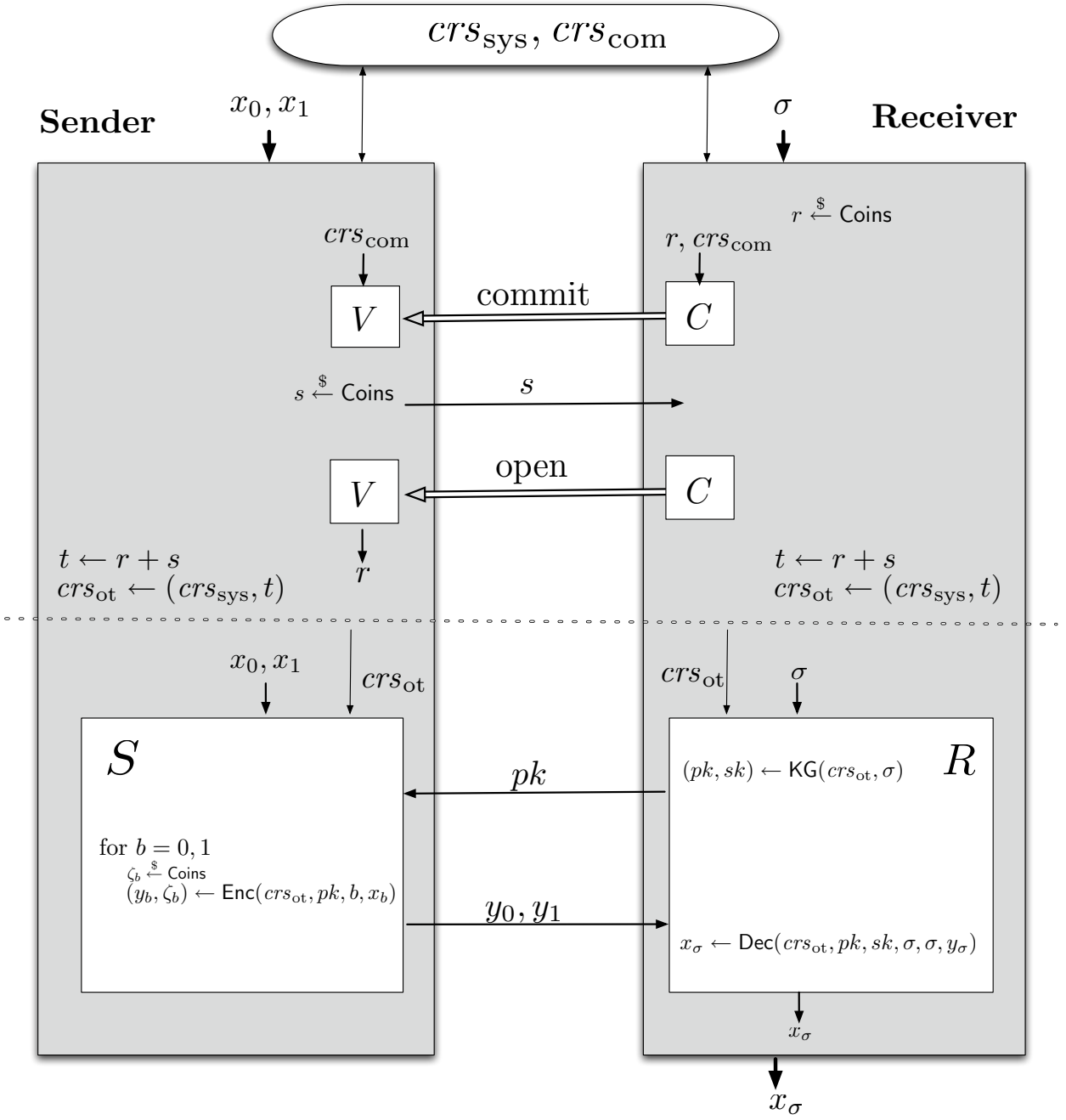


Figure 6: Generic second-corruption adaptively secure OT protocol. Here $\boxed{C} \overset{\text{commit}}{\Longrightarrow} \boxed{V}$ and $\boxed{C} \overset{\text{open}}{\Longrightarrow} \boxed{V}$ denote the commit and the open stages of an adaptive UC-secure commitment protocol based on CRS crs_{com}; and crs_{sys} is the system CRS; and $\boxed{S} \leftrightarrows \boxed{R}$ is the PVW protocol as in Figure 5 but based on the enhanced dual-mode encryption scheme of which KG, Enc, and Dec are the key generation, encryption, and decryption algorithms. Multiple executions can be run based on the same CRS.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Cryptology ePrint Archive, Report 2000/067*, December 2005. Latest version at http://eprint.iacr.org/2000/067/.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002. Full version at http://eprint.iacr.org/2002/059/.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006. Preliminary version appeared in Eurocrypt 2003.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002. Full version at http://eprint.iacr.org/2002/140/.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003. Full version at http://eprint.iacr.org/2002/047/.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *STOC*, pages 426–437. ACM, 2003. Full version at http://www.brics.dk/~jg/STOC03NMcommitment.pdf.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2000.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

[DNO08] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In *Cryptology ePrint Archive, Report 2008/220*, 2008. Available at http://eprint.iacr.org/2008/220/.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GMY04] Juan A. Garay, Philip MacKenzie, and Ke Yang. Efficient and universally composable committed oblivious transfer and applications. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2004.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2007.

[Lin08] Andrew Y. Lindell. Efficient fully-simulatable oblivious transfer. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 52–70. Springer, 2008.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008. Available at http://people.csail.mit.edu/cpeikert/pubs/OTpaper.pdf.

A The Universal Composability Framework

The UC framework was proposed by Canetti for defining the security and composition of protocols [Can01]. In this framework one first defines an "ideal functionality" of a protocol, and then proves that a particular implementation of this protocol operating in a given computational environment securely realizes this ideal functionality. The basic entities involved are n players $P_1, \ldots, P_n$, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The real execution of a protocol π, run by the players in the presence of $\mathcal{A}$ and an environment machine $\mathcal{Z}$, with input z, is modeled as a sequence of *activations* of the entities. The environment $\mathcal{Z}$ is activated first, generating in particular the inputs to the other players. Then the protocol proceeds by having $\mathcal{A}$ exchange messages with the players and the environment. Finally, the environment outputs one bit, which is the output of the protocol.

The security of the protocols is defined by comparing the real execution of the protocol to an ideal process in which an additional entity, the ideal functionality $\mathcal{F}$, is introduced; essentially, $\mathcal{F}$ is an incorruptible trusted party that is programmed to produce the desired functionality of the given task. The players are replaced by dummy players, who do not communicate with each other; whenever a dummy player is activated, it forwards its input to $\mathcal{F}$. Let $\mathcal{A}$ denote the adversary in this idealized execution. As in the real-life execution, the output of the protocol execution is the one-bit output of $\mathcal{Z}$. Now a protocol π *securely realizes* an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-execution adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π in the real-life execution, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal execution. More precisely, if the two binary distribution ensembles, $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{A}$ and players running protocol π (resp., adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$), are computationally indistinguishable (denoted $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$). For further details on the UC framework refer to [Can05].

As was observed in [CKL06], most functionalities cannot be realized in the UC framework without some setup. One common form of setup is the common reference string (CRS). We model a CRS as an ideal functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ which is shown in Figure 7.

Functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

$\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ is parameterized by a PPT sampling algorithm $\mathcal{D}$.

- On input $(\texttt{CRS}, sid)$ from P, verify that $sid = (\mathcal{P}, sid')$ where $\mathcal{P}$ is a set of identities, and $P \in \mathcal{P}$; else ignore the input. Next if there is no value crs recorded then choose $crs \leftarrow \mathcal{D}()$ and record it. Send $(\texttt{CRS}, sid, crs, P)$ to $\mathcal{S}$; when receiving $(\texttt{CRS}, sid, crs)$ from $\mathcal{S}$, send $(\texttt{CRS}, sid, crs)$ to P.

Figure 7: The common reference string ideal functionality, $\mathcal{F}_{\text{CRS}}$.

B Review of Simulatable Public Key Systems from [DN00]

A *simulatable public key system* is defined by a tuple $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ consisting of the key-generation, encryption and decryption algorithms respectively. For the definition of security, we also require the existence of an *oblivious public key generator* $\widetilde{\mathsf{KG}}$ and a corresponding *key-faking algorithm* $\widetilde{\mathsf{KG}}^{-1}$. Similarly, we require an *oblivious ciphertext generator* $\widetilde{\mathsf{Enc}}$ and a corresponding *ciphertext-faking algorithm* $\widetilde{\mathsf{Enc}}^{-1}$. Intuitively, the key-faking algorithm is used to explain a legitimately generated public key as an obliviously generated public key. Similarly, the ciphertext-faking algorithm is used to explain a legitimately generated ciphertext as an obliviously generated ciphertext.

The simulatable pubic key system has the following three properties.

Semantic Security: For all m_0, m_1 in the appropriate domain, consider the experiment $(pk, sk) \leftarrow \mathsf{KG}()$, $C_0 \leftarrow \mathsf{Enc}_{pk}(m_0)$, $C_1 \leftarrow \mathsf{Enc}_{pk}(m_1)$. Then $(pk, m_0, m_1, C_0) \stackrel{c}{\approx} (pk, m_0, m_1, C_1)$.

Oblivious PK Generation: Consider the experiment $(pk, sk) \leftarrow \mathsf{KG}()$, $r \leftarrow \widetilde{\mathsf{KG}}^{-1}(pk)$ and $pk' \leftarrow \widetilde{\mathsf{KG}}(r')$. Then $(r, pk) \stackrel{c}{\approx} (r', pk')$.

Oblivious Cipher-text generation: For any message m in the appropriate domain, consider the experiment $(pk, sk) \leftarrow \mathsf{KG}()$, $C_1 \leftarrow \widetilde{\mathsf{Enc}}_{pk}(r_1)$, $C_2 \leftarrow \mathsf{Enc}_{pk}(m; r_2)$, $r_1' \leftarrow \widetilde{\mathsf{Enc}}_{pk}^{-1}(C_2)$. Then

$$(pk, r_1, C_1) \stackrel{c}{\approx} (pk, r_1', C_2)$$

C Efficiency of Our ℓ-NCE Scheme

Let us look at the efficiency of the construction. For concreteness we will assume that the secure channel used to encrypt the index i are implemented using the NCE protocol of [DN00]. The simulatable public-key system (which we use during channel setup and also which is used in the protocol of [DN00]) can be instantiated with e.g. ElGamal Encryption. Lastly, the symmetric key system can be implemented very efficiently using some variable length encryption algorithm e.g. AES in CBC mode.

The protocol of [DN00] is an *expected* 6 round protocol which exchanges a t bit message using an *expected* $\mathcal{O}(t)$ number of operations.[6] Since we use the NCE protocol to send an index $i \in \{1, \ldots, \ell\}$, this requires $\mathcal{O}(\log \ell)$ number of public key operations. In addition to NCE, we use the simulatable public key system. However, we only use it to perform one encryption/decryption operation and $\mathcal{O}(\ell)$ oblivious key-sampling, ciphertext-sampling operations. For ElGamal, this just means choosing random group elements which is efficient and hence we do not count it as a public key operation. Lastly, for the encryption phase we only use symmetric key operations. This gives us a total of (expected) $\mathcal{O}(\log \ell)$ public key operations, $\mathcal{O}(\ell)$ communication and fewer than (expected) 8 rounds of interaction for the channel setup phase. After channel-setup, encryption is non-interactive and requires only symmetric key operations. However, the encryption of a k bit message requires $\mathcal{O}(\ell k)$ communication.

D Well-Structured Protocols

We need to make sure that the protocol has a well-defined structure so that, when two honest parties execute the protocol, the public information revealed by (even fully) non-committing encryption is fully known in advance. More precisely, this means that the number of rounds, the order of communication (identity of sender/receiver) and the lengths of the exchanged messages should be *completely determined* and fixed for all executions of the protocol. Of course, the protocol execution must depend on some parameters such as the roles of the parties (e.g., who is the sender or receiver in an OT protocol). In general, a protocol π (which does not use setup) defines the behavior of (honest) parties modeled as interactive TMs which accept input from the environment and from each other. We define what it means for such a protocol to be well-formed as follows:

Definition D.1. *A protocol π for the functionality $\mathcal{F}_{\mathrm{SFE}}$ is* well-structured *if, on input $(\mathtt{Input}_I, sid, x_I)$ the initiator I sends some message of length k_0 to R. The responder R stores messages from I until receiving its input $(\mathtt{Input}_R, sid, x_R)$. Then the execution of the protocol proceeds in rounds $i = 1, \ldots, n$ where, in each round i the party b_i sends a message m_i of length k_i to the party $1 - b_i$ (the first message from the initiator I can be considered round 0). The value b_i switches in each round: i.e. P_{b_0} is the initiator and, for each successive round i, $b_i = 1 - b_{i-1}$. We require that the message sizes k_i and the number of rounds n are completely determined by the protocol and are independent of the input values or random coins of the parties.*

Of course, many protocols also rely on some trusted setup modeled as an ideal functionality. We define a *transparent* setup, to be one which does not receive any inputs nor give *any* outputs to the adversary while

[6] For large messages (larger than the security parameter) the protocol of [DN00] can be made *guaranteed* 3 round. However, the tradeoff is that the number of public key operations is at least security parameter. Since we consider very small messages (3 bits) we settle for the *expected* 6 round protocol.

the parties P_0, P_1 are honest. A well-structured protocol may use transparent setup in an arbitrary way. Often, however, protocols must use setup which is not transparent. For example, the CRS ideal functionality (see Figure 7) is usually defined as giving the value of the CRS to the adversary, even when all of the participating parties are honest.

Definition D.2. *A protocol π using some (non-transparent) trusted setup, modeled as an ideal functionality $\mathcal{T}$, is* well-structured *if:*

- *The functionality $\mathcal{T}$ has no special inputs for the adversary $\mathcal{S}$.*
- *In a protocol execution of π with honest parties, the use of the setup $\mathcal{T}$ is such that all side-information given by $\mathcal{T}$ to the adversary is given* before *the parties exchange any messages with each other.*

We call the phase of the protocol starting from when the initiator sends the first message to the receiver, the communication phase. *Prior to that, the adversary might see some information from the use of the setup $\mathcal{T}$. We call this the* setup phase.

For example, in a well-structured protocol using a CRS, the initiator must call the CRS functionality *before* sending the first message to the responder. This ensures that all side-information provided by $\mathcal{T}$ to $\mathcal{S}$ occurs only during the setup phase.

E Proofs

E.1 Proof of Theorem 2.6

We need to show that our protocol in Figure 2 realizes the $\mathcal{F}_{\mathrm{SC}}^{\ell}$ functionality described in Definition 2.5. This functionality is parametrized by two oracles: the message-ignoring oracle $\mathcal{R}$ and the message-processing oracle $\mathcal{M}$. We define these oracles based on our actual protocol construction. In essence the oracle $\mathcal{M}$ corresponds to the actions of the parties for the index i chosen during channel setup, while the oracle $\mathcal{R}$ corresponds to all other indices in $\{1, \ldots, \ell\}$. In particular:

- On input $(\texttt{ChSetup}, sid)$, the oracle $\mathcal{M}$ samples $(pk, sk) \leftarrow \mathsf{KG}()$, $K \leftarrow \mathsf{KG}^{\mathsf{sym}}()$, $C \leftarrow \mathsf{Enc}_{pk}(K)$. The oracle $\mathcal{R}$ samples $pk \leftarrow \widetilde{\mathsf{KG}}()$, $C \leftarrow \widetilde{\mathsf{Enc}}_{pk}()$.
- On input $(\texttt{Send}, sid, m)$, the oracle $\mathcal{M}$ samples $C \leftarrow \mathsf{Enc}_K^{\mathsf{sym}}(m)$. The oracle $\mathcal{R}$ samples C randomly.

We must first show that the oracles $\mathcal{M}$ and $\mathcal{R}$ are indistinguishable. We do so using a hybrid argument.

1. We start with the message-processing oracle $\mathcal{M}$.
2. We now modify the oracle so that, for all `Send` commands, it chooses the symmetric-key ciphertext C randomly instead of computing $C \leftarrow \mathsf{Enc}_K^{\mathsf{sym}}(m)$. This modification is indistinguishable from the initial oracle since ciphertexts produced by the symmetric key scheme are indistinguishable from random ciphertexts.
3. We now modify the oracle from step 2 so that, for all `ChSetup` commands, instead of computing $C \leftarrow \mathsf{Enc}_{pk}(K)$, it computes $C \leftarrow \widetilde{\mathsf{Enc}}_{pk}()$. This oracle is indistinguishable from that of step 2 by the oblivious ciphertext generation property.
4. Lastly we modify the oracle from step 3 so that, for `ChSetup` commands, instead of computing $(pk, sk) \leftarrow \mathsf{KG}()$ it computes $pk \leftarrow \widetilde{\mathsf{KG}}()$. This oracle is indistinguishable from that of step 3 by the oblivious key generation property.

The last step yields the message-ignoring oracle $\mathcal{R}$. Hence $\mathcal{M}$ and $\mathcal{R}$ are computationally indistinguishable.

The two oracle $\mathcal{M}, \mathcal{R}$ now define the complete non-information oracle $\mathcal{N}^\ell$ and hence the ℓ-equivocal secure channel functionality $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}^\ell}$. We must now describe an ideal-world simulation of our ℓ-NCE protocol.

The simulation while both parties are honest is actually very simple since the non-information oracle $\mathcal{N}$ actually runs the protocol and hence there is little that our simulator $\mathcal{S}$ must do! Essentially, to simulate the secure transfer of the index i, the simulator simply sends the length of the message (which is known since ℓ is known) to the adversary $\mathcal{A}$. To simulate the rest of the channel setup phase and also any encryption commands, the simulator $\mathcal{S}$ just passes all information from $\mathcal{N}$ to the adversary $\mathcal{A}$.

The only difficult part is simulating the first corruption. In the ideal world, the simulator is only given the internal state of the single machine $\mathcal{N}_i$ which is the message-processing oracles. In the real-world the state of the corrupted party also includes the randomness for all of the obliviously-generated public keys and ciphertexts (i.e. the internal state of all of the message-ignoring oracles). But the simulator can simulate this easily by using the ciphertext-faking and key-faking algorithms. In other words, for each symmetric key ciphertext C^{sym} produced by an oracle $\mathcal{N}_i$, the simulator simply sets the random coins of the sender as C^{sym} (since it is just a uniformly random value). Moreover, for each public key ciphertext C, the simulator sets the internal state of its sender as $\widetilde{\mathsf{Enc}}_{pk}^{-1}(C)$. Lastly for each public key pk, the simulator sets the internal state of its sender as $\widetilde{\mathsf{KG}}^{-1}(pk)$.

We can view the real world protocol as a series of the same ℓ oracles $\mathcal{N}_1, \ldots, \mathcal{N}_\ell$ where $\mathcal{N}_i$ (for the transferred index i) is the above described message-processing oracle and the rest are message-ignoring oracles. Hence the only difference between the real world and the ideal world is how the random coins of the corrupted party for its message-ignoring oracles are generated. In the real-world the adversary gets the actual coins while in the ideal-world the adversary gets coins produced by the faking algorithms. Let us denote the worlds by tuples showing which oracles are running for all indices other than i (i.e. either message-processing or messsage-ignoring) and how the state is generated (i.e. actual state, faked state). So the real world can be represented as a tuple (message-ignoring, actual). But, by the properties of the simulatable public key system, this is indistinguishable from the world (message-processing, faked). Lastly, since the fake algorithms do not use any secrets, and message-ignoring oracles are indistinguishable from message processing oracles, the world (message-processing, faked) is indistinguishable from (message-ignoring, faked). But this is the ideal world, and hence we have shown that the real-world is indistinguishable from the ideal world.

E.2 Proof of Theorem 2.8

We assume that the underlying protocol π is well-structured and that it realizes the two-party functionality $\mathcal{F}_{\mathrm{SFE}}$ against any second-corruption adaptive adversary. In particular this means that for any second-corruption adaptive adversary $\mathcal{A}_{\mathsf{sec}}$ there exists a simulator $\mathcal{S}_{\mathsf{sec}}$ such that for any environment $\mathcal{Z}_{\mathsf{sec}}$,

$$\mathrm{EXEC}_{\pi,\mathcal{A}_{\mathsf{sec}},\mathcal{Z}_{\mathsf{sec}}} \overset{c}{\approx} \mathrm{EXEC}_{\mathcal{F}_{\mathrm{SFE}},\mathcal{S}_{\mathsf{sec}},\mathcal{Z}_{\mathsf{sec}}}$$

A pictorial illustration can be found in Figure 8 . We wish to construct a simulator $\mathcal{S}$ so that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and π' or with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{SFE}}$, where protocol π' is based on π by applying fully non-committing secure channels $\mathcal{F}_{\mathrm{SC}}$ to protect the communication between the two π-parties P_0 and P_1, and $\mathcal{A}$ denotes fully adaptive adversaries, i.e.,

$$\mathrm{EXEC}_{\pi',\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\mathcal{F}_{\mathrm{SFE}},\mathcal{S},\mathcal{Z}}$$

Without loss of generality, we assume that the adversary $\mathcal{A}$ (resp. $\mathcal{A}_{\mathsf{sec}}$) is just the "dummy" adversary which only follows instructions from the environment $\mathcal{Z}$ (resp. $\mathcal{Z}_{\mathsf{sec}}$). Next we first give the construction of $\mathcal{S}$ based on $\mathcal{S}_{\mathsf{sec}}$, and then demonstrate the validity of $\mathcal{S}$ based on the validity of $\mathcal{S}_{\mathsf{sec}}$.

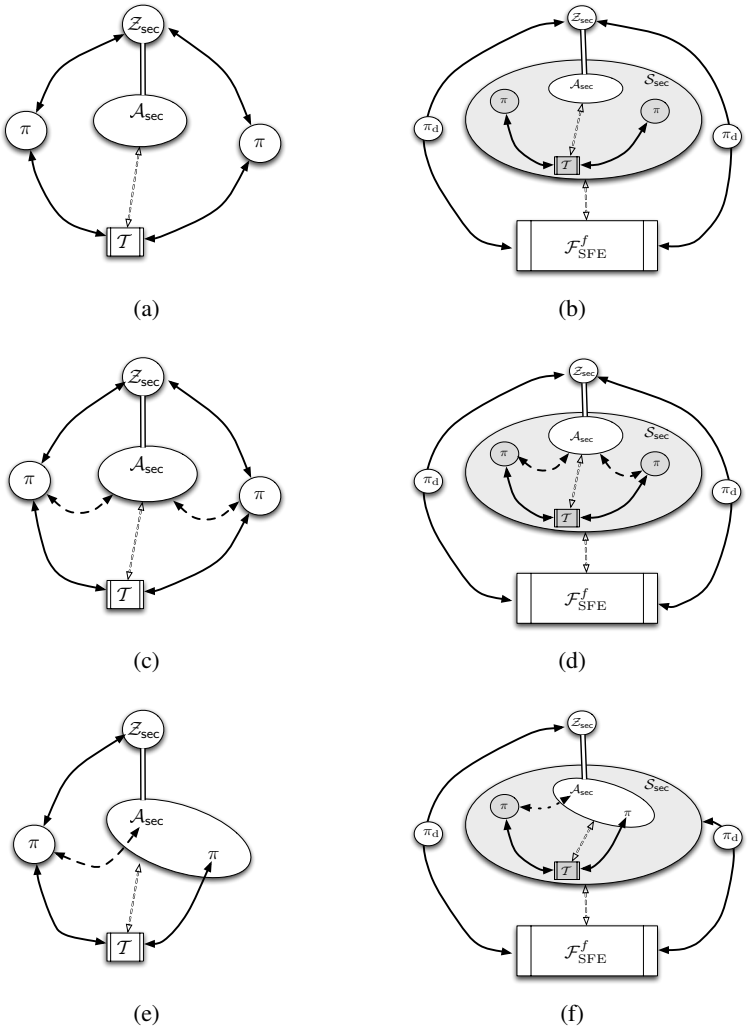


Figure 8: Protocol π under setup $\mathcal{T}$ realized $\mathcal{F}_{\mathrm{SFE}}$ against second-corruption adaptive attacks. Subfigure (a) describes the setup in the real world; whenever a π-party accesses the setup $\mathcal{T}$, some side information could be learned by the adversary $\mathcal{A}_{\mathsf{sec}}$, and after that no extra information will be learned from $\mathcal{T}$; here we consider that $\mathcal{S}_{\mathsf{sec}}$ is a dummy adversary and fully follows environment $\mathcal{Z}_{\mathsf{sec}}$'s instruction, and there is an "absolute" communication channel between $\mathcal{Z}_{\mathsf{sec}}$ and $\mathcal{A}_{\mathsf{sec}}$. Subfigure (b) describes the corresponding ideal world of subfigure (a); whenever the functionality $\mathcal{F}_{\mathrm{SFE}}$ receives an input from a dummy party, it will leak some side information to the ideal world adversary $\mathcal{S}_{\mathsf{sec}}$, and at this moment, $\mathcal{S}_{\mathsf{sec}}$ could simulate the side information from $\mathcal{T}$ to the internal copy of $\mathcal{A}_{\mathsf{sec}}$; further $\mathcal{S}_{\mathsf{sec}}$ learns the trapdoor of the setup. After setup phase $\mathcal{Z}_{\mathsf{sec}}$ needs to make a decision for corrupting parties. Subfigure (c) describes the case that $\mathcal{Z}_{\mathsf{sec}}$ does not corrupt any party; now $\mathcal{A}_{\mathsf{sec}}$ still monitors the communications between the two honest π-parties. Subfigure (d) describes the ideal world of subfigure (c); and $\mathcal{S}_{\mathsf{sec}}$ is evolved from the one in subfigure (b), and it further needs to simulate the communication between two honest π-parties without learning their secret inputs. Subfigure (e) describes the case that $\mathcal{Z}_{\mathsf{sec}}$ corrupts the π-party P_1 (and we always assume the left party is P_0 and the right P_1 in this proof) ; now $\mathcal{A}_{\mathsf{sec}}$ controls the corrupted P_1 immediately after the setup; and $\mathcal{A}_{\mathsf{sec}}$ can interact with honest P_0 in the name of P_1. Subfigure (f) describes the ideal world of subfigure (e); and $\mathcal{S}_{\mathsf{sec}}$ is evolved from the one in subfigure (b); and now $\mathcal{S}_{\mathsf{sec}}$ might extract the secret input used by the corrupted P_1 and send it to the functionality $\mathcal{F}_{\mathrm{SFE}}$ to enable the other party to receive the correct output and to learn some information from the functionality to finish the protocol simulation with the internal copy of $\mathcal{A}_{\mathsf{sec}}$. We note that the case that $\mathcal{Z}_{\mathsf{sec}}$ corrupts P_0 at the very beginning, and the corresponding ideal world can be described similarly as in subfigures (e) and (f).

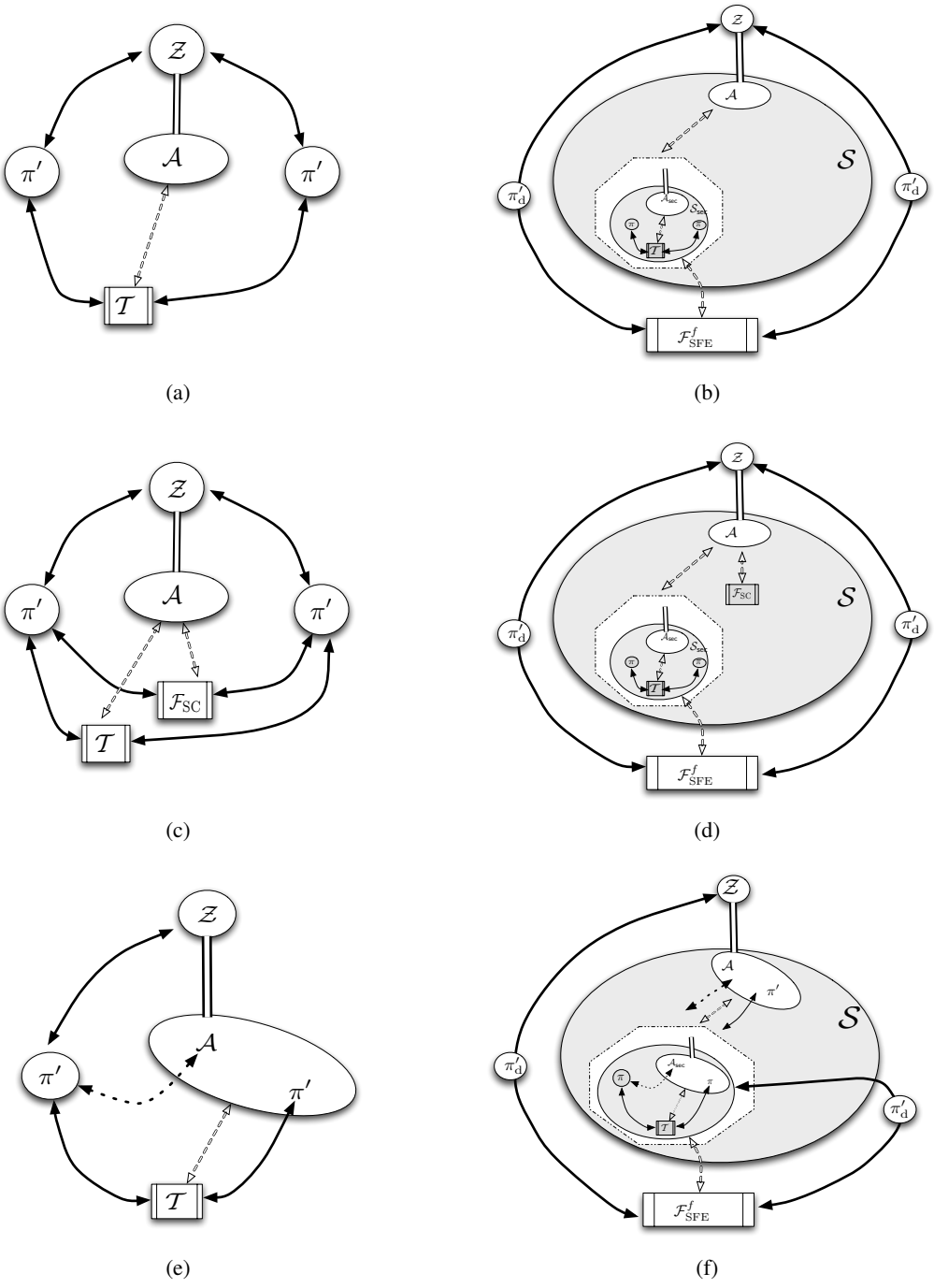


Figure 9: Constructing a simulator $\mathcal{S}$ against fully adaptive attacks based on a simulator $\mathcal{S}_{\mathsf{sec}}$ against second-corruption adaptive attacks. Subfigures (a) and (b) illustrate in the setup stage; subfigures (c) and (d) illustrate the two worlds respectively in which two honest parties communicate with each other through $\mathcal{F}_{\mathrm{SC}}$; while subfigures (e) and (f) illustrate the two worlds respectively in which one party is corrupted and the other party could be corrupted further.

E.2.1 Simulator construction

We define the simulation in terms of the following four stages:

I. The setup phase while both parties are honest.

II. The communication phase while both parties are honest.

III. The first corruption.

IV. Execution after the first corruption.

I. The setup phase while both parties are honest. The simulator $\mathcal{S}$ simulates an internal copy of the (fully adaptive) adversary $\mathcal{A}$ and also the "absolute" communications between the external $\mathcal{Z}$ and the internal $\mathcal{A}$. Please refer to Figure 9(b) . Notice that in the real world (refer to Figure 9(a)), the adversary $\mathcal{A}$ can observe the side information from the setup $\mathcal{T}$ whenever there is a π'-party who is waken up by $\mathcal{Z}$ and accesses to the setup. In order to make a good simulation, $\mathcal{S}$ first initializes an internal copy of $\mathcal{S}_{\mathsf{sec}}$ and simulates the ideal world for $\mathcal{S}_{\mathsf{sec}}$; when $\mathcal{S}$ receives the first message $(\texttt{Input}_I, sid)$ or $(\texttt{Input}_R, sid)$ from $\mathcal{F}_{\mathrm{SFE}}$, it forwards such message to $\mathcal{S}_{\mathsf{sec}}$ and further it plays the role of $\mathcal{Z}_{\mathsf{sec}}$ (through the dummy $\mathcal{A}_{\mathsf{sec}}$) to interacts with $\mathcal{S}_{\mathsf{sec}}$ (a pictorial illustration can be found in octagon in Figure 9(b)); whenever $\mathcal{S}_{\mathsf{sec}}$ supplies the side information of $\mathcal{T}$ to $\mathcal{A}_{\mathsf{sec}}$, the simulator $\mathcal{S}$ returns such information to the internal $\mathcal{A}$. Note that the entire setup of the protocol π is equivalent to the setup of π'.

II. The communication phase while both parties are honest. After setup phase, we consider the case that $\mathcal{Z}$ did not corrupt any party at the very beginning, i.e., $\mathcal{Z}$ did not corrupt any party immediately after the setup up. This stage is evolved from stage (I). Note that now in the real world (please refer to Figure 9(c)), the adversary $\mathcal{A}$ observes that (1) the side information from $\mathcal{T}$, and (2) the side information from $\mathcal{F}_{\mathrm{SC}}$. However $\mathcal{F}_{\mathrm{SC}}$ does not reveal much at this time – just the message lengths and identities of the sender and receiver, and given the protocol π is well-structured, these are known ahead of time. Hence the simulator $\mathcal{S}$ is constructed by continuing the simulator in stage (I) and please also refer to Figure 9(b) (which takes care of (1)) and by further simulating $\mathcal{F}_{\mathrm{SC}}$ to reveal the side information (which takes care of (2)). We give the details below.

Recall that the communication phase proceeds in rounds $i = 0, \ldots, n$ where in each round i the party b_i sends a message of length k_i to party $1 - b_i$. The party b_0 is the initiator, and afterwards $b_i = 1 - b_{i-1}$. If the ideal functionality has delivered a message $(\texttt{Input}, sid, I)$ indicating that the initiator has sent his input, then the simulator $\mathcal{S}$ produces the side information $(\texttt{ChSetup}, sid, I, R)$ and $(\texttt{Send}, I, R, k_0)$ for $\mathcal{A}$ (delivers them one by one) on behalf of $\mathcal{F}_{\mathrm{SC}}$. Once $(\texttt{Input}, sid, R)$ has been received as well, the simulator $\mathcal{S}$ proceeds with rounds $i = 1, \ldots, n$ and in each round i sends the message $(\texttt{Send}, sid, P_{b_i}, k_i)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathrm{SC}}$. When the simulator $\mathcal{S}$ reaches the last real-world message, it sends $(\texttt{Output}, sid, P_0)$ and $(\texttt{Output}, sid, P_1)$ to $\mathcal{F}_{\mathrm{SFE}}$.

We call this the *dummy execution* since the simulator $\mathcal{S}$ does not run any protocol behind the scenes but only passes message lengths to $\mathcal{Z}$. The adversary $\mathcal{A}$ may corrupt a party at any point during this interaction.

III. The first corruption. This includes two evolutions, one from stage (I) and we describe it in the following stage(III-a), and the other is from stage (II) and we describe it in the following stage (III-b).

III-a. We first consider the case that the first corruption occurs at the very beginning and at that moment there was no communication between the two parties. The real world is evolved from the one in stage (I) and please also refer to Figure 9(a). The simulator $\mathcal{S}$ is easier to be constructed since $\mathcal{S}$ is not required to explain the side information from $\mathcal{F}_{\mathrm{SC}}$. Now the $\mathcal{S}$ can be constructed by continuing the one in stage (I) also refer to Figure 9(b); further, whenever $\mathcal{S}$ receives a message from $\mathcal{A}$ (i.e., from the external $\mathcal{Z}$), it plays the role of $\mathcal{Z}_{\mathsf{sec}}$ to send $\mathcal{S}_{\mathsf{sec}}$ that message and obtain the response from $\mathcal{S}_{\mathsf{sec}}$; such response will later be returned to $\mathcal{A}$. Note that here $\mathcal{S}$ forwards the interactions between the internal $\mathcal{S}_{\mathsf{sec}}$ and the external $\mathcal{F}_{\mathrm{SFE}}$. Actually in this case, $\mathcal{S}$ is "equivalent" to $\mathcal{S}_{\mathsf{sec}}$, and $\mathcal{Z}$ is same as $\mathcal{Z}_{\mathsf{sec}}$.

III-b. Next we consider the case that the first corruption occurs when the two parties finish k moves communication, where $k > 0$. The real world is evolved from the one described in stage (II) and refer to Figure 9(c). Now the adversary $\mathcal{A}$ (i.e., $\mathcal{Z}$) has already observed the side information from $\mathcal{F}_{\mathrm{SC}}$ about the first k moves communication, and $\mathcal{S}$ needs to return $\mathcal{A}$ the corresponding k moves communication. The simulator $\mathcal{S}$ is constructed by continuing the one in stage (II), and refer to Figure 9(d) .

After the corruption of a party say P_1, the simulator $\mathcal{S}$ is able to access to P_1's secret input say x_1 from $\mathcal{F}_{\mathrm{SFE}}$, and it continues to play the role of $\mathcal{Z}_{\mathsf{sec}}$ to corrupt P_1 at the very beginning that no communication occurs between the two parties P_0 and P_1; the simulator further plays the role of $\mathcal{Z}_{\mathsf{sec}}$ to simulate a corrupted P_1 based on x_1 and some random coins and interact with $\mathcal{S}_{\mathsf{sec}}$ to produce the first k moves communication messages. These messages will be returned to $\mathcal{A}$. After this point, $\mathcal{S}$ will stop playing the role of $\mathcal{Z}_{\mathsf{sec}}$ to simulate a corrupted P_1 honestly, and it instead plays the role of $\mathcal{Z}_{\mathsf{sec}}$ by feeding $\mathcal{S}_{\mathsf{sec}}$ the messages from the external $\mathcal{Z}$, and returning the responses from $\mathcal{S}_{\mathsf{sec}}$ to $\mathcal{Z}$. Note that here $\mathcal{S}$ forwards the interactions between the internal $\mathcal{S}_{\mathsf{sec}}$ and the external $\mathcal{F}_{\mathrm{SFE}}$. Please refer to Figure 9(f) for a pictorial illustration.

IV. Execution after the first corruption. After the first corruption, whenever the remaining honest party say P_0 is also corrupted, the simulator $\mathcal{S}$ after learning P_0's secret input say x_0 is required to return to P_0's internal state to the external $\mathcal{Z}$. Now the simulator plays the role of $\mathcal{Z}_{\mathsf{sec}}$ to corrupt P_0, and then x_0 will be revealed to $\mathcal{S}_{\mathsf{sec}}$, and further $\mathcal{S}_{\mathsf{sec}}$ will respond with the internal state of P_0 which will be forwarded to $\mathcal{Z}$. This finishes the description of the simulator $\mathcal{S}$.

E.2.2 The validity of the constructed simulator

The simulator $\mathcal{S}$ constructed in the previous subsection is based on $\mathcal{S}_{\mathsf{sec}}$. We still need to demonstrate the validity of $\mathcal{S}$, which can be based on the validity of $\mathcal{S}_{\mathsf{sec}}$. Now assume $\mathcal{S}$ is not valid, which means for any $\mathcal{S}$ there is $\mathcal{Z}$ which can distinguish its interaction with $\mathcal{A}$ and π' from that with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{SFE}}$, we wish to show $\mathcal{S}_{\mathsf{sec}}$ is not valid, i.e., for any $\mathcal{S}_{\mathsf{sec}}$ there is $\mathcal{Z}_{\mathsf{sec}}$ can distinguish its interaction with $\mathcal{A}_{\mathsf{sec}}$ and π from that with $\mathcal{S}_{\mathsf{sec}}$ and $\mathcal{F}_{\mathrm{SFE}}$.

We first construct $\mathcal{S}$ based on any given $\mathcal{S}_{\mathsf{sec}}$ as in the previous subsection; based on the assumption that $\mathcal{S}$ is not valid, we can have an environment $\mathcal{Z}$ which can distinguish the two worlds with non-negligible probability ϵ; further, based on $\mathcal{Z}$, we give the construction of $\mathcal{Z}_{\mathsf{sec}}$ to distinguish the two world with probability ϵ' which is also non-negligible.

Before we go to the details of the construction of $\mathcal{Z}_{\mathsf{sec}}$, we note that the $\mathcal{S}$ in the previous subsection based on $\mathcal{S}_{\mathsf{sec}}$ is carefully designed, because all $\mathcal{Z}$'s possible strategies are covered. $\mathcal{Z}$ could distinguish the two world based on: 1, just setup (I), or by after setup immediately corrupting both parties (I,IV); 2, by corrupting just one party immediately after setup (I, III-a), or further corrupting the other party (I, III-a,IV); 3a, by corrupting no party immediately after setup, but corrupting only one party in the communication stage (I, II, III-b), or further corrupting the other party (I, II, III-b, IV); 3b, by corrupting no party in the whole execution (I, II).

Based on the above discussion of $\mathcal{Z}$'s strategies, we describe our construction of $\mathcal{Z}_{\mathsf{sec}}$.

1. $\mathcal{Z}_{\mathsf{sec}}$ runs an internal copy of $\mathcal{Z}$ and simulates the execution with $\mathcal{Z}$. At the very beginning, the adversary $\mathcal{A}$ (which is controlled by $\mathcal{Z}$) could expect to learn the leaking information from $\mathcal{T}$; the environment $\mathcal{Z}_{\mathsf{sec}}$ obtain such information for $\mathcal{A}$ by controlling $\mathcal{A}_{\mathsf{sec}}$ to learn it from $\mathcal{T}$; this is illustrated in Figure 10(a). Here we also include the case that both π'-parties are corrupted by $\mathcal{Z}$ at this very beginning; now $\mathcal{Z}_{\mathsf{sec}}$ just corrupts the two π-parties.

2. In the case that one π'-party is corrupted by $\mathcal{Z}$ at the very beginning, $\mathcal{Z}_{\mathsf{sec}}$ corrupts the corresponding π-party; if $\mathcal{Z}$ sends any input to the other honest π'-party, $\mathcal{Z}_{\mathsf{sec}}$ just forwards the corresponding honest π-party; if $\mathcal{Z}_{\mathsf{sec}}$ obtains any output from the honest π-party, $\mathcal{Z}_{\mathsf{sec}}$ will play the role of the honest π'-party to return the output to the internal $\mathcal{Z}$; any message from the honest π-party to $\mathcal{A}_{\mathsf{sec}}$ and will be forwarded to $\mathcal{A}$, and any

message from $\mathcal{A}$ to the honest π'-party will also be forwarded by $\mathcal{A}_{\mathsf{sec}}$ to the honest π-party. This is the easy case and is illustrated in Figure 10(b).

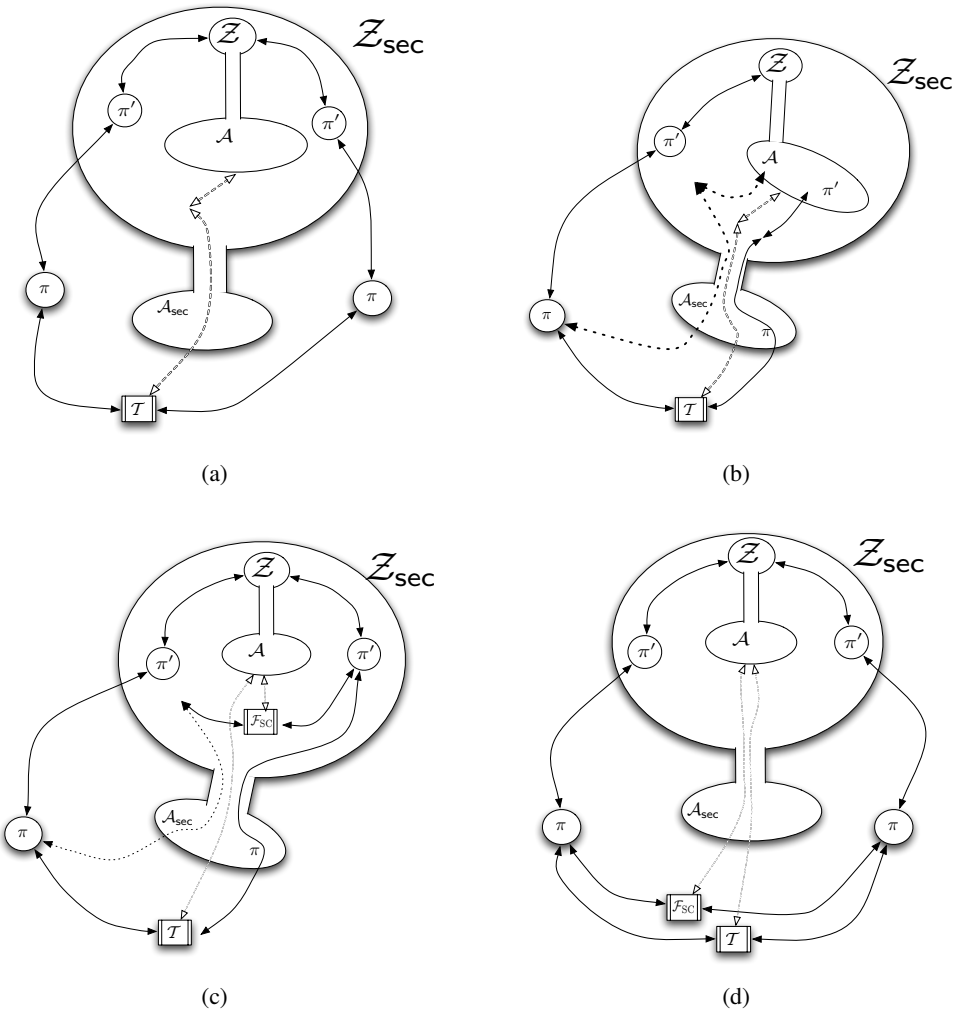


Figure 10: Constructing environment $\mathcal{Z}_{\mathsf{sec}}$ based on environment $\mathcal{Z}$.

3. Next we go to the interesting case that $\mathcal{Z}$ does not corrupt any party at this very beginning; $\mathcal{Z}$ could corrupt one party after k moves communication between the two honest parties where $k > 0$, or it even corrupts two parties, or else it corrupts no party during the whole execution. Since the environment $\mathcal{Z}_{\mathsf{sec}}$ has no idea of this important further corruption information, it just makes a decision that with probability p_0 it corrupts party P_0, and with p_1 corrupts party P_1, and with p' corrupts no party, where we let p_0, p_1, p' be non-negligible and $p_0 + p_1 + p' = 1$; if the decision is "wrong" and the following corruption information made by $\mathcal{Z}$ is inconsistent with $\mathcal{Z}_{\mathsf{sec}}$'s prediction, then $\mathcal{Z}_{\mathsf{sec}}$ just halts.

3a. The case that $\mathcal{Z}_{\mathsf{sec}}$ corrupts P_1 at the very beginning is illustrated in Figure 10(c). The honest π-party P_0 behaviors in the same way as that in Case 2. The corrupted π-party P_1 now is controlled by $\mathcal{A}_{\mathsf{sec}}$; whenever the honest π'-party P_1 produces a message for the other honest π'-party P_0, $\mathcal{A}_{\mathsf{sec}}$ plays the role of the corrupted π-party P_1 to send that message to the honest π-party P_0; similarly when the honest π-party P_0 sends a message

to the corrupted π-party P_1, that message will be forwarded to the internal π'-party P_1. The case that $\mathcal{Z}_{\mathsf{sec}}$ corrupts P_0 at the very beginning can be illustrated in the very similar way.

3b. The case that $\mathcal{Z}_{\mathsf{sec}}$ corrupts no party is illustrated in Figure 10(d), where $\mathcal{Z}_{\mathsf{sec}}$ just forwards the inputs/outputs between $\mathcal{Z}$ and the honest parties, and $\mathcal{A}_{\mathsf{sec}}$ forwards the leaking information from $\mathcal{T}$ and from $\mathcal{F}_{\mathrm{SC}}$ to $\mathcal{A}$.

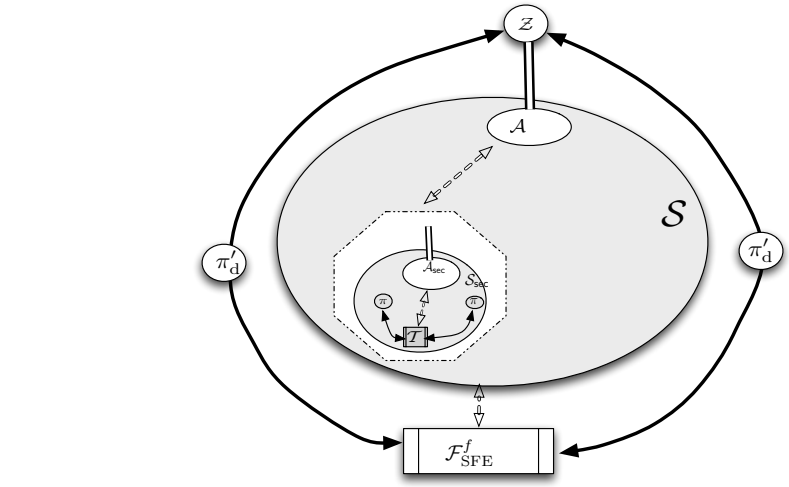
Consider that $\mathcal{Z}$ distinguishes the two worlds in case 1 with probability α, and in case 2 with probability β, and in case 3a by corrupting party P_0 at the very beginning with probability γ_0, and in case 3a by corrupting party P_1 at the very beginning with probability γ_1, and in case 3b by corrupting no party with probability γ'. Note that $\mathcal{Z}$ can distinguish the two world with non-negligible probability, which means at least one of $\alpha, \beta, \gamma_0, \gamma_1, \gamma'$ is non-negligible.

Further note that in case 1 and 2, and also case 3 when $\mathcal{Z}_{\mathsf{sec}}$ made a correct decision, if $\mathcal{Z}_{\mathsf{sec}}$ interacts with $\mathcal{S}_{\mathsf{sec}}$ and $\mathcal{F}_{\mathrm{SFE}}$, the view of the internal $\mathcal{Z}$ has the same distribution as the view of $\mathcal{Z}$ interacting with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{SFE}}$; and similarly, if $\mathcal{Z}_{\mathsf{sec}}$ interacts with $\mathcal{A}_{\mathsf{sec}}$ and π protocol, the view of the internal $\mathcal{Z}$ has the same distribution as that of its interacting with $\mathcal{A}$ and π' protocol. Therefore, $\mathcal{Z}_{\mathsf{sec}}$ can distinguish the two worlds with probability $\epsilon' = \alpha + \beta + p_0\gamma_0 + \ p_1\gamma_1 + p'\gamma$. Recall that at least one e_i is non-negligible, and recall that all p_0, p_1, p' are non-negligible, we can conclude that ϵ' is non-negligible, which means the two worlds can be distinguished by $\mathcal{Z}_{\mathsf{sec}}$ with non-negligible probability. This completes the proof.

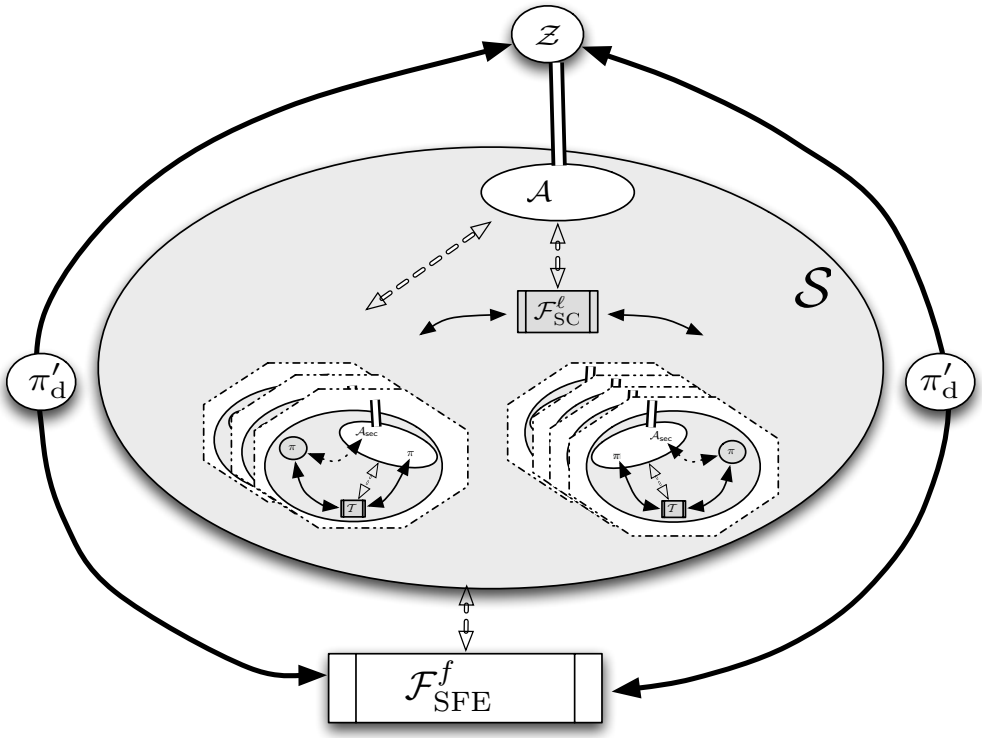
E.3 Proof sketch of Theorem 2.9

The simulation and the proof are very similar to that of Theorem 2.8 as presented above. So here we just give proof ideas. In the pervious simulator construction, at the very beginning, $\mathcal{S}$ runs $\mathcal{S}_{\mathsf{sec}}$ to simulate the side information from $\mathcal{T}$; in this simulator construction, $\mathcal{S}$ will also use $\mathcal{S}_{\mathsf{sec}}$ to simulate the side information from $\mathcal{T}$. However the side information from the secure channel functionality is much harder to be simulated since $\mathcal{F}_{\mathrm{SC}}^{\ell}$ leaks much more information than that from $\mathcal{F}_{\mathrm{SC}}$; this bring major difficulty for our construction of this simulator. In the previous simulation, $\mathcal{F}_{\mathrm{SC}}$ just leaks the length and direction of the protocol communication messages as well as the identities and roles of the involved parties; in this simulation, $\mathcal{F}_{\mathrm{SC}}^{\ell}$, beyond the side information from $\mathcal{F}_{\mathrm{SC}}$, also reveals the encrypted valid communication in a meaningful channel among several noise channels. Recall that now the inputs/outputs from both parties are very short; this enables the simulator to guess the real inputs/outputs for one party and explore the power of $\mathcal{S}_{\mathsf{sec}}$ to prepare the every possible valid communication between the parties, and further explore the indistinguishability of the meaningful channel and noise channels to fill each channel with valid communication. We give the pictorial illustration in Figure 11(a) and Figure 11(b). First $\mathcal{S}$ simulates $\mathcal{S}_{\mathsf{sec}}$ to obtain the leaking information from $\mathcal{T}$ as demonstrated in Figure 11(a). Then $\mathcal{S}$ continues the simulation of $\mathcal{S}_{\mathsf{sec}}$ in multiple ways by feeding a tuple (b, x, y) which means P_b is corrupted at the very beginning with input x and expects output y. Now $\mathcal{S}_{\mathsf{sec}}$ fissions into ℓ copies and each copy corresponds to a tuple (b, x, y); this is demonstrated in Figure 11(b).

The validity argument is also very similar to that in the previous proof. Now in case 3a in the previous proof, $\mathcal{Z}_{\mathsf{sec}}$ make a decision to behavior based on (b, x, y) with non-negligible probability $p_{b,x,y}$. Note that in the previous proof $\mathcal{Z}_{\mathsf{sec}}$ makes a decision to behavior based on b with non-negligible probability p_b. As a result, $\mathcal{Z}_{\mathsf{sec}}$ can distinguish the two worlds with non negligible probability, instead of previous $\epsilon' = \alpha + \beta + p_0\gamma_0 + p_1\gamma_1 + p'\gamma'$, here $\epsilon' = \alpha + \beta + \sum_{b,x,y} p_{b,x,y}\gamma_{b,x,y} + p'\gamma'$. Note that that among $\alpha, \beta, \{\gamma_{b,x,y}\}, \gamma'$ at least one is non-negligible, and we can set all $p_{b,x,y}$'s and p' are non-negligible. This completes the proof idea.



(a)



(b)

Figure 11: Constructing simulator $\mathcal{S}$ based on $\mathcal{S}_{\mathsf{sec}}$.

E.4 Proof of Theorem 3.2

To finish the proof, we first need to construct a simulator such that there is no PPT environment $\mathcal{Z}$ can distinguish the execution with the adversary $\mathcal{A}$ and protocol $\hat{\pi}_{\mathrm{OT}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world (the multi-session version of the protocol from Figure 6), and the execution with the simulator $\mathcal{S}$ and ideal functionality $\hat{\mathcal{F}}_{\mathrm{OT}}$, where $\mathcal{A}$ is only allowed to corrupt parties in the second-corruption adaptive way. Next we give the construction of the simulator. Note that the underlying commitment is an adaptively secure commitment; so the simulator can take advantage of the extractability and equivocality of the UC commitment. Note also that the underlying enhanced dual mode encryption is secure, the simulator can use the $\mathsf{FakeEnc}$ and Recons algorithms and DualKG algorithm for the simulation.

- **Simulating the communication with $\mathcal{Z}$.** The simulator simulates the adversary $\mathcal{A}$ internally which can interact with the external environment $\mathcal{Z}$, i.e., whenever $\mathcal{A}$ and $\mathcal{Z}$ exchange messages with each other, the simulator will forward such messages between them.

- **Trusted setup.** The simulator computes the CRS $(crs_{\mathrm{sys}}, crs_{\mathrm{com}})$ with the trapdoor $(\tau_{\mathrm{sys}}, \tau_{\mathrm{com}})$. When parties query $\mathcal{F}_{\mathrm{CRS}}$, return $(\mathtt{CRS}, sid, \langle crs_{\mathrm{sys}}, crs_{\mathrm{com}}\rangle)$. Note that the CRS can be learned by $\mathcal{A}$ even no party is corrupted, and no mode information has been committed by the simulator in the trusted setup stage.

- **Simulation of no corruption case.** It is the easy case and ignored.

- **Simulation of the initially corrupted receiver case.** Given the adversary $\mathcal{A}$ is the second-corruption adaptive adversary, here we consider the case that at the very beginning the receiver is corrupted while the sender is honest; the sender can be corrupted at any point in the communication stage. The simulator chooses the messy mode, and simulates the sender as follows.

  First, in the coin-tossing phase, the simulator chooses t such that the messy trapdoor τ_{ot} is known for $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, t)$; then the simulator computes s such that $t = r + s$, where r is extracted from the commitment value from a corrupted receiver. Further, in the transfer phase, the simulator can obtain the sender's input $x_{1-\rho}$ for the non-messy branch $1-\rho$ by querying the functionality $\hat{\mathcal{F}}_{\mathrm{OT}}$ with the input bit $1-\rho$, i.e., the simulator in the name of corrupted receiver sends $(\mathtt{Receiver}, sid, ssid, 1-\rho)$ to the functionality and obtain $(\mathtt{Output}, sid, ssid, x_{1-\rho})$ from the functionality; note that here ρ is extracted by using the messy trapdoor τ_{ot} from the pk computed by the corrupted receiver; then the simulator computes $y_{1-\rho}$ honestly for $x_{1-\rho}$ by using randomly selected $\zeta_{1-\rho}$, and computes y_ρ by running the fake encryption algorithm, i.e., $(y_\rho, \omega_\rho) \leftarrow \mathsf{FakeEnc}(crs_{\mathrm{ot}}, \tau_{\mathrm{ot}}, pk, \rho, \rho)$. Later if the sender is corrupted, then the simulator based on the learned x_ρ runs the internal state reconstruction algorithm to compute ζ_ρ, i.e., $\zeta_\rho \leftarrow \mathsf{Recons}(crs_{\mathrm{ot}}, \tau_{\mathrm{ot}}, pk, \rho, \rho, y_\rho, \omega_\rho, x_\rho)$, and returns (ζ_0, ζ_1) as the sender's internals.

- **Simulation of the initially corrupted sender case.** Here we consider the case that at the very beginning the sender is corrupted while the receiver is honest; the receiver can be corrupted at any point in the communication stage. The simulator chooses the decryption mode and simulates the receiver as follows.

  First, in the coin-tossing phase, the simulator computes a fake commitment value; then the simulator chooses t such that the decryption trapdoor τ_{ot} is known for $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, t)$; after obtaining s, the simulator computes r such that $t = r + s$, where s is received from a corrupted sender; later the simulator equivocates the previous fake commitment value into r. Further, in the transfer part, the simulator runs $(pk, sk_0, sk_1) \leftarrow \mathsf{DualKG}(crs_{\mathrm{ot}}, \tau_{\mathrm{ot}})$. After receiving ciphertexts (y_0, y_1), the simulator computes (x_0, x_1) for the functionality $\mathcal{F}_{\mathrm{OT}}$ as the sender's input by applying the decryption algorithm $x_b \leftarrow \mathsf{Dec}(crs_{\mathrm{ot}}, pk, sk_b, b, b, y_b)$ for $b = 0, 1$. Later if the receiver is corrupted the simulator, based on the learned σ, reveals sk_σ as the receiver's internals.

To finish the proof, we still need to argue that in the case that the receiver is corrupted at the beginning and in the case that the sender is corrupted at the beginning, no PPT environment can distinguish the two worlds.

First we consider the corrupted receiver case. We define several hybrid experiments and argue that the difference between them are negligible.

HYB1 This is the CRS hybrid world.

HYB2 Change the CRS setup, and the simulator knows τ_{com}. Now the simulator, instead of randomly selecting s, extracts r from c first based on the extractability of the commitment, and compute s based on r and t where t is part of the CRS for the transferring phase in the messy mode.

HYB1 and HYB2 are indistinguishable given the underlying commitment is UC secure.

HYB3 Change the CRS setup further, and the simulator knows τ_{sys}. Now the simulator can know the trapdoor τ_{ot} for the CRS $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, t)$ in the messy mode.

HYB2 and HYB3 are indistinguishable given the mode indistinguishability of the underlying enhanced dual mode encryption.

HYB4 Instead of honestly producing the ciphertexts (y_0, y_1) for both branches, the simulator extracts the messy branch number and uses the faking encryption to produce the ciphertext for the messy branch, and later runs the state reconstruction algorithm to compute the internals when the sender is further corrupted. The simulator still deals with the non-messy branch honestly.

HYB3 and HYB4 are indistinguishable given the messy branch identification and ciphertext equivocation property of the underlying enhanced dual mode encryption.

Note that HYB4 is exactly the ideal world. Therefore the difference between the real and the ideal worlds are negligible.

Next we consider the corrupted sender case. We again show that the difference between the two worlds based on a hybrid argument.

HYB1 This is the CRS hybrid world.

HYB2 Similar to the proof above. The simulator changes the CRS setup, and knows τ_{com}. Now the simulator, instead of honestly produce c, uses the fake committing algorithm, and after learning s, equivocate c to make t be part of the CRS for the transferring phase in the decryption mode.

HYB1 and HYB2 are indistinguishable given the underlying commitment is UC secure.

HYB3 Similar to the proof above. Change the CRS setup further, and the simulator knows τ_{sys}. Now the simulator can know the trapdoor τ_{ot} for the CRS $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, t)$ in the decryption mode.

Again HYB2 and HYB3 are indistinguishable given the mode indistinguishability of the underlying enhanced dual mode encryption.

HYB4 Instead of honestly producing the pk, the simulator use the dual key generation to produce pk and sk_0, sk_1, later reveals the corresponding decryption key as its internal state when the receiver is further corrupted.

HYB3 and HYB4 are indistinguishable given the encryption key duality property of the underlying enhanced dual mode encryption.

Note that HYB4 is exactly the ideal world. Therefore the difference between the real and the ideal worlds are negligible.

Based on the above argument, we conclude that the multi-session version of the protocol in Figure 6 is second-corruption adaptively secure.

F QR-Based Enhanced Dual Mode Encryption

We first review the dual mode encryption based on quadratic residuosity (QR) assumption in [PVW08], then we show this scheme is actually an *enhanced* dual mode encryption as defined in Definition 3.1 under the same assumption.

The QR-based dual mode encryption in [PVW08] is based on a variant of Cocks' encryption scheme [Coc01] as follows. For $N \in \mathbb{N}$, let $\mathbb{J}_N$ denote the set of all $x \in \mathbb{Z}_N$ with Jacobi symbol $+1$, and $\mathbb{QR}_N \subset \mathbb{J}_N$ denote the set of all quadratic residues in $\mathbb{Z}_N^*$, and $(\frac{t}{N})$ denote the Jacobi symbol of t in $\mathbb{Z}_N^*$. The message space is $\{\pm 1\}$.

- The key generation algorithm $(pk, sk) \leftarrow \mathsf{CocKG}(1^\lambda)$: Randomly select two λ-bit primes p and q and set $N \leftarrow pq$. Randomly select $r \xleftarrow{\$} \mathbb{Z}_N^*$ and set $y \leftarrow r^2$. Set $pk \leftarrow (N, y)$, and $sk \leftarrow r$. Output (pk, sk).
- The encryption algorithm $(c, s) \leftarrow \mathsf{CocEnc}(pk, m)$: Parse pk as (N, y). Randomly select $s \xleftarrow{\$} \mathbb{Z}_N^*$ such that $(\frac{s}{N}) = m$, and compute $c \leftarrow s + y/s$. Output (c, s).
- The decryption algorithm $m \leftarrow \mathsf{CocDec}(pk, sk, c)$: Parse pk as (N, y). Parse sk as r. Compute the Jacobi symbol of $c + 2r$, i.e., $m \leftarrow (\frac{c+2r}{N})$. Output m.

We next present the enhanced dual mode cryptosystem which is based on the above scheme.

- The parameter generation algorithm for the messy mode and the decryption mode are presented as follows.
  - $\star$ $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mathrm{mes})$: Randomly select two λ-bit primes p and q and set $N \leftarrow pq$. Randomly select $y \xleftarrow{\$} \mathbb{J}_N \setminus \mathbb{QR}_N$. Set $crs \leftarrow (N, y)$ and $\tau \leftarrow (p, q)$. Output (crs, τ).
  - $\star$ $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mathrm{dec})$: Randomly select two λ-bit primes p and q and set $N \leftarrow pq$. Randomly select $s \xleftarrow{\$} \mathbb{Z}_N^*$ and set $y \leftarrow s^2 \bmod N$. Set $crs \leftarrow (N, y)$ and $\tau \leftarrow s$. Output (crs, τ).
- The key generation algorithm $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$:

  Parse the crs as (N, y). Randomly select $r \xleftarrow{\$} \mathbb{Z}_N^*$, set $sk \leftarrow r$ and $pk \leftarrow r^2/y^\sigma$. Output (pk, sk).
- The encryption algorithm $(c, \zeta) \leftarrow \mathsf{Enc}(crs, pk, b, m)$:

  Parse the crs as (N, y). Set $pk_b \leftarrow (N, pk \cdot y^b)$. Compute $(c, \zeta) \leftarrow \mathsf{CocEnc}(pk_b, m)$. Output (c, ζ).
- The decryption algorithm $m \leftarrow \mathsf{Dec}(crs, pk, sk, b, \sigma, c)$:

  Parse the crs as (N, y). Set $pk_b \leftarrow (N, pk \cdot y^b)$. If $b = \sigma$, then compute $m \leftarrow \mathsf{CocDec}(pk_b, sk, c)$. Output m.
- The messy branch identification algorithm $\rho \leftarrow \mathsf{MessyId}(crs, \tau, pk)$:

  Parse the crs as (N, y) where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p, q) where $N = pq$. If $pk \in \mathbb{QR}_N$, then set $b \leftarrow 1$; otherwise set $b \leftarrow 0$. Output b.
- The fake encryption algorithm $(c, \omega) \leftarrow \mathsf{FakeEnc}(crs, \tau, pk, b, \rho)$:

  Parse the crs as (N, y) where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p, q) where $N = pq$. If $b = \rho$, then set $y' \leftarrow pk \cdot y^b$ and set $pk_b \leftarrow (N, y')$; note that $y' \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Compute $(c, \zeta_0) \leftarrow \mathsf{CocEnc}(pk_b, m)$, i.e., and compute $c \leftarrow \zeta_0 + y'/\zeta_0$. Based on the messy characterization explored in Lemma 6.1 in [PVW08], compute $\zeta_1, \zeta_2, \zeta_3$ such that $\zeta_1 = \zeta_0 \bmod p$ and $\zeta_1 = y'/\zeta_0 \bmod q$, $\zeta_2 = y'/\zeta_0 \bmod p$ and $\zeta_2 = \zeta_0 \bmod q$, $\zeta_3 = y'/\zeta_0 \bmod p$ and $\zeta_3 = y'/\zeta_0 \bmod q$; note that two of $(\frac{\zeta_i}{N})$ are $+1$, and the other two are -1. Set $\omega \leftarrow (\zeta_0, \zeta_1, \zeta_2, \zeta_3)$. Output (c, ω).

- The internal state reconstruction algorithm $\zeta \leftarrow \mathsf{Recons}(crs, \tau, pk, b, \rho, c, \omega, m)$:

  In the case that $b = \rho$, set $\zeta \leftarrow \zeta_i$ where ζ_i is from ω such that $(\frac{\zeta_i}{N}) = m$. Output ζ.

- $(pk, sk_0, sk_1) \leftarrow \mathsf{DualKG}(crs, \tau)$:

  Randomly select $r \xleftarrow{\$} \mathbb{Z}_N^*$, set $sk_b \leftarrow r \cdot s^b$ for $b \in \{0, 1\}$, and $pk \leftarrow r^2$. Output (pk, sk_0, sk_1).

Theorem F.1. *Under the quadratic residuosity assumption, the above scheme is an enhanced dual mode encryption as defined in Definition 3.1.*

Proof sketch. The proof is very similar to that of Theorem 6.2 in [PVW08]. The first two properties can be argued directly based on their proof. The fourth property can also be argued based on their proof because in the key generation, the randomness used by the KG is exactly the decryption information which is included in decryption key sk. For the third property, besides their argument, we need further to show for all $m \in \{\pm 1\}$, the distribution of (c, ζ) from the honest encryption algorithm is identical to that produced by the fake encryption and reconstruction algorithms.

For $m = +1$, the former distribution can be written as $\{(c, \zeta) | \zeta \xleftarrow{\$} \mathbb{J}_N, c = \zeta + y/\zeta\}$; the latter distribution can be written as $\{(c, \zeta_0) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, \zeta_1 = y/\zeta_0 \bmod q\}$ which can be further rewritten as $\{(c, \zeta_0) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0\}$; so the two distribution are identical.

For $m = -1$, the former distribution can be written as $\{(c, \zeta) | \zeta \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta + y/\zeta\}$; the latter distribution can be written as $\{(c, \zeta_1) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, \zeta_1 = y/\zeta_0 \bmod q\}$ which can be further rewritten as $\{(c, \zeta_1) | \zeta_0 \xleftarrow{\$} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, (\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N})\}$; note that $(\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N}) = -1 \cdot (\frac{\zeta_0}{N})$ since $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$; given $\zeta_0 \xleftarrow{\$} \mathbb{J}_N$, we can have $\zeta_1 \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N$; therefore the latter distribution can be rewritten as $\{(c, \zeta_1) | \zeta_1 \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, (\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N})\}$, and then $\{(c, \zeta_1) | \zeta_1 \xleftarrow{\$} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta_1 + y/\zeta_1\}$ which is identical to the former distribution.

Together we show the two distributions are identical, which concludes that the third property is also satisfied. This completes the proof. □