Somewhat Non-Committing Encryption and Efficient Adaptively Secure Oblivious Transfer

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

April 15, 2009

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 *semi-adaptive* security which is slightly stronger than static security but *significantly weaker than fully adaptive security*. The main difference between adaptive and semi-adaptive security is that, for semi-adaptive security, the simulator is not required to handle the case where *both* parties start out honest and one becomes corrupted later on during the protocol execution. As such, semi-adaptive security is much easier to achieve than fully adaptive security. We then give a simple, generic protocol compiler which transforms any semi-adaptively 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 semi-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 in 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 semi-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 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, adaptively secure and composable OT protocol. In particular, to transfer an n-bit message, we use a constant number of rounds and $O(n)$ public key operations.

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

[*]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.

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, KO04, GMY04, JS07, LP07, IPS08, Lin09]. 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, Lin09] provide UC/concurrent adaptive security but only in the erasure model, and [DN03] provides UC adaptive security but only for an honest majority, and [KO04] do not allow for an adversary that eventually corrupts *all* parties. 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. Lastly, we mention the work of [CDD$^+$04], which gives a generic compiler from static to adaptive security using secure channels. Unfortunately, this compiler does not provide full adaptive security (does not allow for post-execution corruptions) and, as was noted in [Lin09], crucially relies on rewinding and hence cannot be used in the UC framework.

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 indeed seems to be the case. The recent work of [LZ09], proves a (black-box) separation between enhanced trapdoor permutations (which allow for static OT) and adaptively secure OT, showing that the latter is indeed "more complex" in a theoretical sense. This complexity is reflected in practice as well. 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 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. Interestingly, Ishai *et al.* [IKLP06] give the first OT protocol constructions against malicious corruptions *without* using zero knowledge proofs; this result was later strengthened in [Hai08]. Two very recent and efficient concrete protocols not using zero-knowledge proofs are given in [PVW08, Lin08]. 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.

[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.

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 has been no known way to *combine* these techniques.

1.1 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 *semi-adaptive* security which is slightly stronger than static security but significantly weaker than fully adaptive security. In particular, a semi-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 semi-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 semi-adaptive security and the problem of realizing secure channels. We note that a similar compiler was studied in [CDD$^+$04]. As we mentioned, that compiler only works in the stand-alone setting and transforms a statically secure protocol into one which is adaptively secure *without* post-execution corruptions. In contrast, our protocol works in the UC setting, and results in fully adaptive security, but requires the starting protocol to be semi-adaptively secure (a new notion which we formally define later).

Unfortunately, we saw that the construction of secure-channels is a difficult problem 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 semi-adaptively secure protocol for a task with small input/output domains (such as bit-OT) into a fully adaptively secure protocol. We also show that this methodology can, in special cases, be applied to tasks with larger domain sizes such as string-OT with long strings.

We apply our methodology to the OT protocol of Peikert *et al.* [PVW08], resulting in the first efficient and adaptively secure OT protocols. 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 semi-adaptively secure. We then employ our compiler, using somewhat non-committing encryption, to convert them into (fully) adaptively UC-secure OT protocols.

1.2 Concurrent and independent work

Following the line of work of [IKLP06, Hai08], the recent result of [CDMW09] gives a generic black-box compiler from semi-honest adaptively secure OT to fully malicious adaptively secure OT, using cut-and-choose techniques. Although the end result of our work is the same (adaptively secure OT), the two works take very different approaches which complement each other well: the compiler of [CDMW09] transforms semi-honest + adaptive security into malicious + adaptive security in the special case of OT, while our compiler is a general transformation from malicious + semi-adaptive security to malicious + adaptive security. The two starting notions of security (semi-honest + adaptive vs. malicious + semi-adaptive) are incomparable and thus both compilers are useful in different scenarios. In particular, our compiler can be used in conjunction with the OT protocol of [PVW08] and results in an extremely efficient adaptively-secure OT protocol using a constant number of rounds and $O(n)$ public-key operations to transfer an n-bit string.[2] In contrast, the compiler of [CDMW09] shows how to base adaptively-secure OT on a simulatable cryptosystem in a black-box way, but at the expense of running $\Omega(\lambda^2)$ copies of the underlying semi-honest OT protocol, where λ is the security parameter, and thus requiring $\Omega(\lambda^2 n)$ operations for n-bit OT. Therefore our protocol can be significantly more efficient.

To avoid diluting the main ideas of the paper, we put all proofs in the appendix, together with background material, efficiency considerations, our enhanced version of the QR dual-mode cryptosystem, and our DDH version of adaptively secure bit- and string-OT.

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, simulating the first corruption becomes much easier 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! To formalize this, we will define a notion of semi-adaptive security where the simulator needs to be able to simulate static corruptions as well as the case where one party starts out corrupted and the other party becomes corrupted later on (but *not* the case where *both* parties start out honest and may become corrupted later). The formal notion (with some additional restrictions imposed on the simulator) appears in Section 2.4. Informally, we have argued the following claim:

Claim 2.1. *(Simplified and incorrect version) Assume that a two-party protocol π for a task $\mathcal{F}$ is semi-adaptively secure. Then the protocol is also fully adaptively secure if all communication between the parties is sent over an idealized secure channel.*

The above claim (when formalized and corrected) will already allow us to compile many protocols for tasks like OT into adaptively secure ones. However, we would like to improve the efficiency of this compilation. Unfortunately, idealized secure channels are hard to achieve physically and implementing such channels cryptographically 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 2.1 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.

[2]Technically, if one thinks of n as a function of λ, we require $O(\max(\lambda, n))$ operations.

Finally, we show that somewhat non-committing encryption is often *good enough* to transform a semi-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. The idea of realizing secure channels against adaptive adversaries as part of a compilation of adaptively secure protocols goes back to [BH92] where this is done in the erasure model. Several non-committing encryption schemes in the non-erasure model 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.[3]

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 $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$

The ideal functionality $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$ interacts with an initiator I and a receiver R. It consists of a channel-setup phase, after which the two parties can send arbitrarily many messages from one to another. The functionality is parameterized by a non-information oracle $\mathcal{N}$. It 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, stop the execution of $\mathcal{N}$ and give the adversary $\mathcal{S}$ 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, $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$.

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 $\mathcal{F}_{\text{SC}}^{\mathcal{N}}$ 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.

Definition 2.2. *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 $\texttt{ChSetup}$, $\texttt{Corrupt}$ commands, produces no output. We call the function-*

[3]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.

ality $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}^{\mathrm{full}}}$*, or just* $\mathcal{F}_{\mathrm{SC}}$ *for brevity, a* (fully) non-committing secure channel. *A real-world protocol which realizes* $\mathcal{F}_{\mathrm{SC}}$ *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 2.3. *A machine* $\mathcal{R}$ *is called a* message-ignoring *oracle if, on any input* $(\mathtt{Send}, sid, P, m)$*, it ignores the value* m *and processes only the input* $(\mathtt{Send}, sid, P, |m|)$*. 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 2.4. *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* $i \overset{\$}{\leftarrow} \{1, \ldots, \ell\}$*. 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* $(\mathtt{ChSetup}, sid, P)$ *or* $(\mathtt{Send}, sid, P, m)$*, 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* $(\mathtt{Corrupt}, sid, P)$*, 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* $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}^\ell}$ *an* ℓ-equivocal non-committing secure channel. *For brevity, we will also use the notation* $\mathcal{F}_{\mathrm{SC}}^{\ell}$ *to denote* $\mathcal{F}_{\mathrm{SC}}^{\mathcal{N}^\ell}$ *for some such oracle* $\mathcal{N}^\ell$*. Lastly, a real world protocol which realizes* $\mathcal{F}_{\mathrm{SC}}^{\ell}$ *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 $\mathcal{F}_{\mathrm{SC}}^{\ell}$, 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 $\mathcal{F}_{\mathrm{SC}}^{\ell}$. 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 scheme 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.

Theorem 2.5. *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.*

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 R while I stores sk_i.
3. The receiver 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 $m \leftarrow \mathsf{Dec}^{\mathsf{sym}}_K(C_i)$.

Figure 2: Construction of an ℓ-NCE protocol.

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 two-party SFE

As an application of ℓ-NCE, we give a general theorem, along the lines of Claim 2.1, showing that a protocol with semi-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 give a formal definition of semi-adaptive security. The main part of the definition is to look at corruption strategies which are more restricted than fully adaptive ones, but less so than static ones.

Definition 2.6. *An adversarial strategy is second-corruption adaptive if* either *at least one of the parties is corrupted prior to protocol execution* or *no party is ever corrupted. In the former case, the other party can be adaptively corrupted at any point during or after protocol execution. In other words, the first corruption (if it occurs) must be static and the second corruption can then be adaptive.*

Intuitively, we'd like to say that a protocol is semi-adaptively secure if it is secure with respect to second-corruption adaptive strategies. Unfortunately, there are two subtleties that we must consider. 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 we require that, when trusted setup is used, the semi-adaptive simulator simulates this setup independently of which party is corrupted. We call this property setup-adaptive simulation.

Definition 2.7. *A simulator $\mathcal{S} = (\mathcal{S}^{\mathrm{setup}}, \mathcal{S}^{\mathrm{prot}})$ is setup-adaptive if it proceeds by first running $\mathcal{S}^{\mathrm{setup}}$ to simulate all trusted setup and then running $\mathcal{S}^{\mathrm{prot}}$ (which is given any output generated by $\mathcal{S}^{\mathrm{setup}}$) to simulate the protocol execution. Moreover $\mathcal{S}^{\mathrm{setup}}$ does not get to see which parties are corrupted (but $\mathcal{S}^{\mathrm{prot}}$ does).*

The second subtlety comes from the following type of problem. As we outlined in our informal discussion, we wish to run the semi-adaptive simulator once the first party gets corrupted and then "lie" that the simulated conversation took place over the secure channel. However, when the first party gets corrupted after the protocol execution, then the ideal functionality has already computed the outputs using the honest inputs and will therefore not accept anymore inputs from the semi-adaptive simulator. Recall that we run the semi-adaptive simulator with respect to an adversary $\mathcal{A}$ which follows the protocol execution using the corrupted party's honest input x. If the semi-adaptive simulator extracts the same input x as the one used by $\mathcal{A}$, then we also know the corresponding output and can give it to the semi-adaptive simulator on behalf of the ideal functionality. Therefore it is crucial that the semi-adaptive simulator can only submit the actual input x. We call this property *input-preserving*.

Definition 2.8. *We say that an adversary $\mathcal{A}$ is* protocol-honest *if it corrupts one of the parties P prior to protocol execution and then follows the honest protocol specification using some input x on behalf of the corrupted party. A simulator $\mathcal{S}$ is input-preserving if, during the simulation of a protocol-honest adversary that corrupts P and runs the honest protocol with input x, the simulator $\mathcal{S}$ submits the* same *input x to the ideal functionality $\mathcal{F}^f_{\text{SFE}}$ on behalf of P.*

Putting Definition 2.6, Definition 2.7 and Definition 2.8 together, we are finally ready to define semi-adaptive security.

Definition 2.9. *We say that a protocol π semi-adaptively realizes the ideal functionality $\mathcal{F}$ if there exists a setup-adaptive and input-preserving PPT simulator $\mathcal{S}$ such that, for any PPT adversary $\mathcal{A}$ and environment $\mathcal{Z}$ which follow a second-corruption adaptive adversarial strategy, we have $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.*

Lastly, we define the notion of a *well-structured* protocol. Since even non-committing encryption *commits* the simulator to the lengths of the exchanged messages, the number of such messages, and the identities of the sender and receiver of each message, we require 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 inputs or random tape of the participating parties. Almost all known constructed protocols for cryptographic tasks are well-structured and any protocol can be easily converted into a well-structured protocol. The formal definition of well-structured protocols is relegated to Appendix D.

Functionality $\mathcal{F}^f_{\text{SFE}}$

The functionality $\mathcal{F}^f_{\text{SFE}}$ 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 $\langle I, x_I\rangle$ and send the message $(\texttt{Input}_I, sid)$ to the adversary $\mathcal{S}$. Ignore future $(\texttt{Input}_I, \ldots)$ inputs. Similarly, upon receiving the input value $(\texttt{Input}_R, sid, x_R)$ from the responser R, record the value $\langle R, x_R\rangle$ and send the message $(\texttt{Input}_R, sid)$ to the adversary $\mathcal{S}$. Ignore future $(\texttt{Input}_R, \ldots)$ inputs.

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

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

We now have all the definitions needed to formally state the correctness of our compilers for transforming a semi-adaptively secure protocol into a (fully) adaptively secure protocol. First we look at the simple compiler using idealized secure channels. We now state the corrected version of Claim 2.1.

Theorem 2.10. *Let $\mathcal{F}^f_{\text{SFE}}$ 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 $\mathcal{F}^f_{\text{SFE}}$ is semi-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 2.11. *Let $\mathcal{F}_{\text{SFE}}^{f}$ be the two-party ideal functionality computing some 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 $\mathcal{F}_{\text{SFE}}^{f}$ is semi-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 2.11 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, following the modeling of [Can05].

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 $x_i \in \{0, 1\}$, 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.

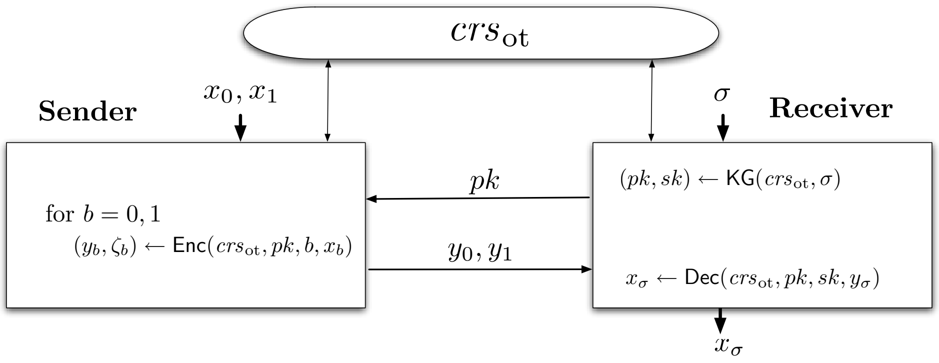
Figure 4: The oblivious transfer ideal functionality, $\mathcal{F}_{\text{OT}}$.

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.

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



crs_{ot}
Sender
x_0, x_1
σ
Receiver
for $b = 0, 1$
$(y_b, \zeta_b) \leftarrow \mathsf{Enc}(crs_{\mathrm{ot}}, pk, b, x_b)$
pk
y_0, y_1
$(pk, sk) \leftarrow \mathsf{KG}(crs_{\mathrm{ot}}, \sigma)$
$x_\sigma \leftarrow \mathsf{Dec}(crs_{\mathrm{ot}}, pk, sk, y_\sigma)$
x_σ

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

$(b = 0)$ 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.

3.2 Semi-adaptively secure OT

In order to make the PVW OT protocol adaptively secure using our methodology, we need to make it semi-adaptively secure (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 crs_{ot} based on which party is corrupt – i.e. the CRS should be in messy mode to handle a corrupt receiver or in decryption mode to handle a corrupt sender. This is a problem for us since the definition of semi-adaptive security requires that the simulator is setup-adaptive which means that it must simulate the CRS independently of any information on which parties are corrupted. We solve this issue by using a coin-tossing protocol to choose the CRS of the PVW OT protocol. Of course, coin-tossing requires the use of a UC secure commitment scheme which also need their own CRS (crs_{com})! However, if we use an (efficient) adaptively secure commitment scheme (e.g. [DN02, DG03]) then the simulator's choice of crs_{com} can be independent on which party is corrupted. Unfortunately, this approach only works if the CRS for the OT protocol comes from a uniform distribution (over some group) and this too is not the case in all instantiations of the PVW protocol. However, we observe that the CRS of the OT protocol (crs_{ot}) can be divided into two parts $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$, where a *system CRS* crs_{sys} can be independent of which party is corrupted (i.e. can be the same for both messy and decryption mode) but may not be uniform, while crs_{tmp} determines the mode (messy or decryption) and thus needs to depend on which party is corrupted, but this part is required to be uniform. Therefore we can use an ideal CRS functionality to choose the setup for our protocol which consists of $(crs_{\mathrm{com}}, crs_{\mathrm{sys}})$ and then run a coin-flipping protocol to choose the uniform value crs_{tmp}.

Secondly, we must now consider the cases where one party is corrupted from the beginning, but the second party becomes corrupted adaptively during the protocol execution. Let us first consider the case where the sender starts out corrupted. 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. Now, 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 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 $\{0,1\}^n$ 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., compute $(G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}}) \leftarrow \mathsf{PG}_{\mathrm{sys}}(1^\lambda)$ and $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(\mu, G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$ where G is a group with operator "+", and set $crs \leftarrow (crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$ and $\tau \leftarrow (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$. Note that the system CRS is independent of mode μ.
- $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$. The key generation algorithm KG is a randomized algorithm which takes crs and a key type $\sigma \in \{0,1\}$, and outputs a key pair (pk, sk), where pk is an encryption key and sk the corresponding decryption key for message encrypted on key type σ. 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 outputs a ciphertext c for a message m on encryption type b. Here ζ is the random coins used for encryption.
- $m \leftarrow \mathsf{Dec}(crs, pk, sk, c)$. The decryption algorithm Dec is a deterministic algorithm which decrypts ciphertext c into plaintext m.
- $\rho \leftarrow \mathsf{MessyId}(crs, \tau, pk)$. The messy branch identification algorithm $\mathsf{MessyId}$ is a deterministic algorithm which based on the trapdoor τ computes the key type ρ corresponding to a messy branch of pk.
- $(c, \omega) \leftarrow \mathsf{FakeEnc}(crs, \tau, pk, \rho)$. The fake encryption algorithm $\mathsf{FakeEnc}$ is a randomized algorithm. For the messy branch ρ, the ciphertext c is faked by using the trapdoor τ, and some internal information ω is saved for reconstructing the random coins used for encryption.
- $\zeta \leftarrow \mathsf{Recons}(crs, \tau, pk, \rho, c, \omega, m)$. The internal state reconstruction algorithm Recons is a deterministic algorithm. When the plaintext m is supplied for the faked ciphertext c in messy branch ρ, the algorithm recover the used random coins ζ based on previously generated internal information ω.
- $(pk, sk_0, sk_1) \leftarrow \mathsf{DualKG}(crs, \tau)$. The dual key generation algorithm DualKG is a randomized algorithm, which based on the trapdoor τ, outputs an encryption key pk, and two decryption keys sk_0, sk_1 corresponding to key type 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 $\mu \in \{0,1\}$, $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mu)$, $m \in \{0,1\}^n$, $\sigma \in \{0,1\}$, and $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$, the decryption on branch σ is correct except with negligible probability; i.e.,* $\mathsf{Dec}(crs, pk, sk, c) = m$*, 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 (ii) the two tem-*

poral CRSes are computationally indistinguishable from random elements in group G, i.e.,

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

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* $m \in \{0,1\}^n$*,* $\mathsf{Enc}(crs, pk, \rho, \cdot)$ *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 semi-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}, while the second piece is for an adaptively secure UC commitment protocol 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 crs_{tmp} based on s and r. The temporal CRS crs_{tmp} 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 (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 protocol in Figure 6 semi-adaptively realizes* $\mathcal{F}_{\mathrm{OT}}$ *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 semi-adaptively secure OT. For example, good candidates for adaptively secure UC commitments can be found in [DN02, DG03], while a QR-based dual-mode encryption scheme is presented in [PVW08]. In Appendix F, we show that this scheme also satisfies Definition 3.1. As mentioned in Section 1, a semi-adaptively secure OT protocol can also be 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. (See Appendix G for more details.)

3.3 Efficient and adaptively secure Bit OT protocol

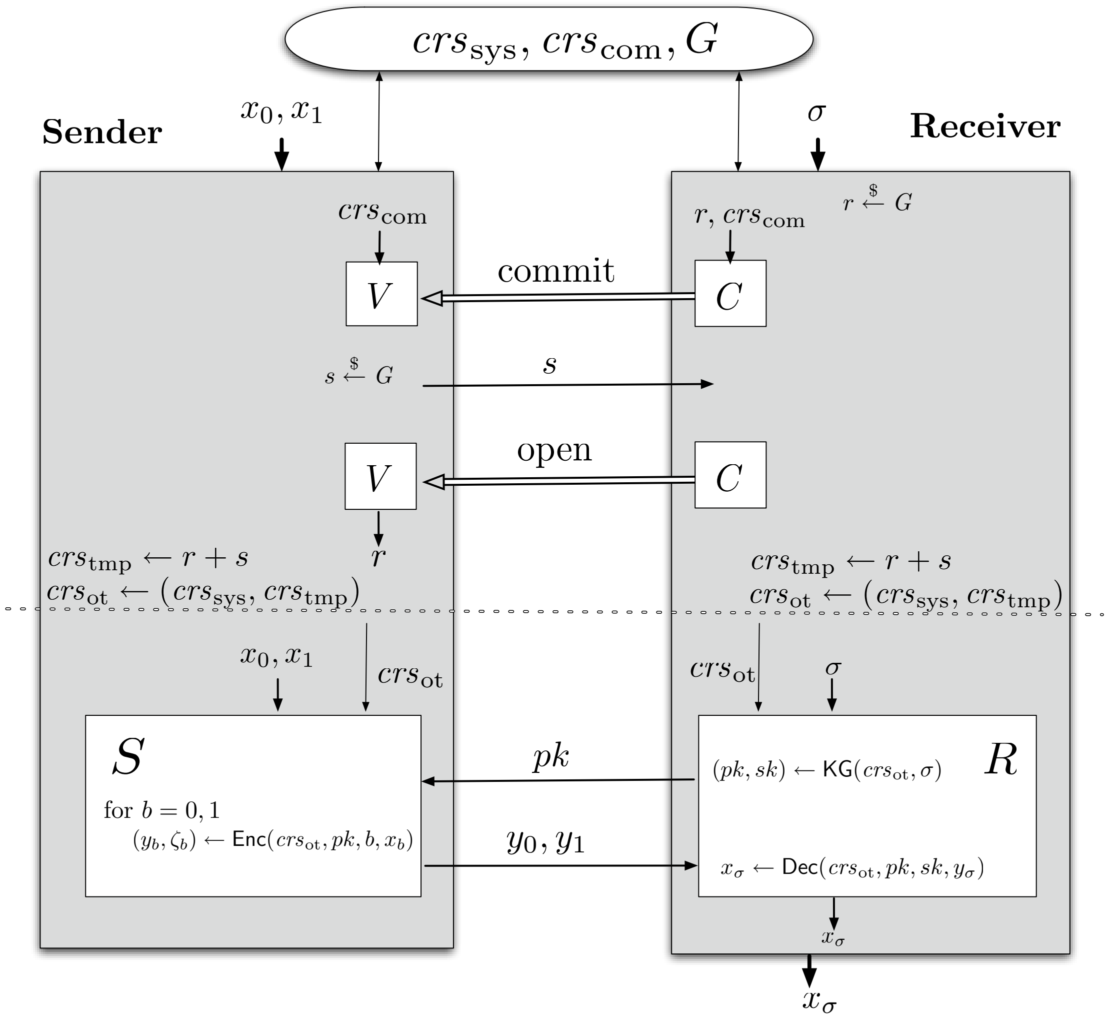
We now apply our compiler from Section 2.4 to the protocol in Figure 6, to immediately obtain an efficient adaptively secure OT protocol in the UC framework.

Corollary 3.3. *Assume that the DDH, QR, and DCR assumptions hold. Then there exists an adaptively secure protocol that UC-realizes the* bit-*OT functionality* $\mathcal{F}_{\mathrm{OT}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid world, running in (expected) constant number of rounds and using (expected) constant number of public-key operations.*

Justification for the assumptions is as follows: efficient adaptive UC commitments can be realized in the CRS model under the DCR assumption [DN02], non-committing and somewhat non-committing encryption can be constructed under DDH ([DN00] and Section 2, respectively), while enhanced dual-model encryption exists under the QR assumption ([PVW08] and Section 3).

3.4 Efficient and adaptively secure *string* OT

In Appendix G we show how to instantiate our framework using the DDH version of PVW. We also show how to then efficiently implement string OT in Appendix G.3. Although this does not follow our framework in a strict sense, it does rely on the use of somewhat non-committing encryption and the properties of the PVW protocol. The resulting efficiency is captured by the following theorem.



$crs_{\text{sys}}, crs_{\text{com}}, G$
Sender
x_0, x_1
σ
Receiver
crs_{com}
r, crs_{com}
$r \overset{\$}{\leftarrow} G$
V
commit
C
$s \overset{\$}{\leftarrow} G$
s
V
open
C
r
$crs_{\text{tmp}} \leftarrow r + s$
$crs_{\text{ot}} \leftarrow (crs_{\text{sys}}, crs_{\text{tmp}})$
$crs_{\text{tmp}} \leftarrow r + s$
$crs_{\text{ot}} \leftarrow (crs_{\text{sys}}, crs_{\text{tmp}})$
x_0, x_1
crs_{ot}
crs_{ot}
σ
S
for $b = 0, 1$
$(y_b, \zeta_b) \leftarrow \mathsf{Enc}(crs_{\text{ot}}, pk, b, x_b)$
pk
$(pk, sk) \leftarrow \mathsf{KG}(crs_{\text{ot}}, \sigma)$
R
y_0, y_1
$x_\sigma \leftarrow \mathsf{Dec}(crs_{\text{ot}}, pk, sk, y_\sigma)$
x_σ
x_σ

Figure 6: Generic semi-adaptively secure OT protocol. Here $\boxed{C} \xrightarrow{\text{commit}} \boxed{V}$ and $\boxed{C} \xrightarrow{\text{open}} \boxed{V}$ denote the commit and the open stages of an adaptive UC-secure commitment protocol based on CRS crs_{com}. crs_{sys} is the system CRS, and $\boxed{S} \leftrightarrows \boxed{R}$ is the PVW protocol (Figure 5), but based on our enhanced dual-mode encryption scheme.

Theorem 3.4. *Assume that the DDH and DCR assumptions hold. Then there exists an adaptively secure protocol that UC-realizes the* string-*OT functionality $\mathcal{F}_{\text{OT}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid world, and can transfer an n-bit string in (strict) constant number of rounds and using (strict) $O(n)$ public-key operations.*

Acknowledgements

We thank Ran Canetti, Yevgeniy Dodis, Yuval Ishai, 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.

[BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In *EUROCRYPT*, pages 307–323, 1992.

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

[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/`.

[CDD$^+$04] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *J. Cryptology*, 17(3):153–207, 2004.

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2009.

[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.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, pages 418–430, 2000.

[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.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer, 2002. Full version at `http://www.brics.dk/RS/01/41/BRICS-RS-01-41.pdf`.

[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 Pil Joong Lee and Jung Hee Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 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.

[Hai08] Iftach Haitner. Semi-honest to malicious oblivious transfer – the black-box way. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2008.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In *STOC*, pages 99–108. ACM, 2006.

[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.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2004.

[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.

[Lin09] Yehuda Lindell. Adaptively secure two-party computation with erasures. In *CT-RSA*, 2009. Available at `http://eprint.iacr.org/2009/031/`. To appear.

[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.

[LZ09] Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 183–201. Springer, 2009.

[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.

[Nie03] Jesper Buus Nielsen. On protocol security in the cryptographic model. *Dissertation Series DS-03-8, BRICS*, 2003. `http://www.brics.dk/DS/03/8/BRICS-DS-03-8.pdf`.

[Nie05] Jesper Buus Nielsen. Universally composable zero-knowledge proof of membership. *Manuscript*, 2005. Available at `http://www.daimi.au.dk/~buus/n05.pdf`.

[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 $P_1, \dots, P_n$, 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 $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$). 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 $(pk, m_0, m_1, C_0) \overset{c}{\approx} (pk, m_0, m_1, C_1)$.

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 $(r, pk) \overset{c}{\approx} (r', pk')$.

Oblivious Cipher-text generation: For any message m in the appropriate domain, consider the experiment $(pk, sk) \leftarrow \mathsf{KG}(), C_1 \leftarrow \widetilde{\mathsf{Enc}}_{pk}(r_1), C_2 \leftarrow \mathsf{Enc}_{pk}(m; r_2), r'_1 \leftarrow \widetilde{\mathsf{Enc}}^{-1}_{pk}(C_2)$. Then

$$(pk, r_1, C_1) \overset{c}{\approx} (pk, r'_1, C_2)$$

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.[4] 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

Definition D.1. *A protocol π for the functionality $\mathcal{F}^{f}_{\mathrm{SFE}}$ 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 $1 - b_i$ where b_i switches in each round. In other words, the first message from the initiator I can be considered round 0 so that 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.

E Proofs

E.1 Proof of Theorem 2.5

We need to show that our protocol in Figure 2 realizes the $\mathcal{F}^{\ell}_{\mathrm{SC}}$ functionality described in Definition 2.4. 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 $(\mathtt{ChSetup}, sid)$, 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 $(\mathtt{Send}, sid, m)$, the oracle $\mathcal{M}$ samples $C \leftarrow \mathsf{Enc}^{\mathsf{sym}}_{K}(m)$. 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.

[4] 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.

2. We start with the message-processing oracle $\mathcal{M}$.
3. We now modify the oracle so that, for all `Send` commands, it chooses the symmetric-key ciphertext C randomly instead of computing $C \leftarrow \mathsf{Enc}^{\mathsf{sym}}_K(m)$. This modification is indistinguishable from the initial oracle since ciphertexts produced by the symmetric key scheme are indistinguishable from random ciphertexts.
4. 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.
5. 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 $\mathcal{F}^{\mathcal{N}^\ell}_{\mathrm{SC}}$. 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 $\widetilde{\mathsf{Enc}}^{-1}_{pk}(C)$. 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 $\mathcal{N}_1, \dots, \mathcal{N}_\ell$ 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 2.10

By assumption, the underlying protocol π is well-structured and there is a simulator $\mathcal{S}_{\mathsf{semi}}$ which is setup-adaptive and input-preserving, such that π realizes the two-party functionality $\mathcal{F}^f_{\mathrm{SFE}}$ against a second-corruption adaptive adversary. In particular for any second-corruption adaptive adversary $\mathcal{A}_{\mathsf{sec}}$ there is $\mathcal{S}_{\mathsf{semi}}$ such that for any environment $\mathcal{Z}_{\mathsf{sec}}$

$$\mathrm{REAL}_{\pi,\mathcal{A}_{\mathsf{sec}},\mathcal{Z}_{\mathsf{sec}}} \overset{c}{\approx} \mathrm{IDEAL}_{\mathcal{F}^f_{\mathrm{SFE}},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}}$$

We now construct a simulator $\mathcal{S}$ for the protocol π' that uses fully non-committing secure channels. The simulator $\mathcal{S}$ runs an internal copy of the (fully adaptive) adversary $\mathcal{A}$ attacking the protocol π'. We can assume, without loss of generality, that the adversary $\mathcal{A}$ is just the "dummy" adversary which only follows instructions from the environment $\mathcal{Z}$. 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. First our simulator $\mathcal{S}$ initializes a copy of the semi-adaptive simulator $\mathcal{S}_{\mathsf{semi}} = (\mathcal{S}^{\text{setup}}_{\mathsf{semi}}, \mathcal{S}^{\text{prot}}_{\mathsf{semi}})$. For stage (I), the entire setup phase of the protocol π is simulated by $\mathcal{S}^{\text{setup}}_{\mathsf{semi}}$ which is oblivious of which parties are corrupted (recall Definition 2.7).

II. The communication phase while both parties are honest. The main idea behind this simulation is that the real-world adversary does not see much at this time – just the message lengths and identities of the sender and receiver. However, because the protocol π is well-structured, these are known ahead of time. Hence the simulator $\mathcal{S}$ can simulate this phase by simply forwarding this publicly known data. 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 $b_i = 1 - b_{i-1}$. 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 $\mathcal{F}_{\text{SC}}$. 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. Assume that the first corruption is that of the party P_{first}. Then the simulator $\mathcal{S}$ gets the entire content of the ideal-world view of P_{first} including its input x_{first} from the environment and (possible) outputs y_{first} from $\mathcal{F}^f_{\text{SFE}}$ depending on the point at which the corruption occurs.

The simulator $\mathcal{S}$ then produces an *imagined execution* between P_0 and P_1 as follows. It constructs a machine $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ which is a (new) environment that corrupts P_{first} immediately after the setup phase but honestly uses the inputs specified by $\mathcal{Z}$ to run the protocol π on behalf of P_{first}. The simulator $\mathcal{S}$ then activates the machine $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$ (passing it any output from $\mathcal{S}^{\text{setup}}_{\mathsf{semi}}$) and runs the simulation by $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$ for the environment $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ (the adversary $\mathcal{A}$ is always dummy w.l.o.g. and hence we do not specify it). It runs this simulation, for as many rounds of the communication phase as occurred *before* P_{first} was corrupted.

In addition, $\mathcal{S}$ acts as the ideal functionality $\mathcal{F}^f_{\text{SFE}}$ for $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$. If $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$ attempts to give input to the ideal functionality then, since $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$ is input-preserving (recall Definition 2.8), this input *is* x_{first} and $\mathcal{S}$ just ignores this. If $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$ asks for output (at a point after both inputs have been submitted) then $\mathcal{S}$ passes it the actual output y_{first} on behalf of $\mathcal{F}^f_{\text{SFE}}$.

Once $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ reaches the point where the environment $\mathcal{Z}$ corrupted P_{first}, the simulator $\mathcal{S}$ takes the internal state of $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}$ (i.e. the randomness used to run the protocol on behalf of the party P_{first} and the view of the protocol execution) and sets it as the internal state of P_{first} to be given to the actual environment $\mathcal{Z}$. This corresponds to *patching* the dummy execution by claiming that the imagined execution took place over the secure channel.

IV. Execution after the first corruption. After the first corruption, the simulator $\mathcal{S}^{\text{prot}}_{\mathsf{semi}}$ is left to simulate the execution without interference.

Indistinguishability of Simulation: Now we need to argue the indistinguishability of the simulation. Intuitively, this follows simply from the fact that $\mathcal{S}_{\mathsf{semi}}$ can simulate a second-corruption adaptive adversary. In the following arguments we assume that the environment $\mathcal{Z}$ corrupts some party at some point. If not then in the ideal world as well as the real world, the environment $\mathcal{Z}$ only sees the lengths of the messages defined by the protocol π so the real world and ideal world are indistinguishable. If a corruption does occur at some point, then we use the following hybrid argument for a series of indistinguishable games.

Game 1 – The Ideal World: We define Game 1 to be $\text{IDEAL}_{\mathcal{F}^f_{\text{SFE}}, \mathcal{S}, \mathcal{Z}}$ - that is the ideal-world simulation with the environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ (as described above) interacting with the ideal functionality $\mathcal{F}$.

Game 2 – Second Corruption Adaptive Ideal World: In the simulation, the imagined execution is generated after the first party is corrupted *ex post facto*. We now define a new execution in which the imagined execution is the one that executes all along.

More precisely, we define an environment $\mathcal{Z}_{\mathsf{sec}}$, which runs an internal copy of $\mathcal{Z}$. However $\mathcal{Z}_{\mathsf{sec}}$ chooses a bit $b \leftarrow \{0,1\}$ randomly and corrupts the party P_b prior to protocol execution. The environment $\mathcal{Z}_{\mathsf{sec}}$ gets the input x_b for P_b from $\mathcal{Z}$, and runs a the protocol for P_b honestly using the input x_b and randomness r_b. In addition $\mathcal{Z}^b_{\mathsf{sec}}$ passes the message lengths, and identities of sender and receiver to $\mathcal{Z}$.

If $\mathcal{Z}$ requests the corruption of a party P_{first} then, if $\mathsf{first} \neq b$ the environment $\mathcal{Z}_{\mathsf{sec}}$ gives the output 0 and quits. Else, if $\mathsf{first} = b$, give the randomness r_b and the actual messages produced and received on behalf of the party P_b to the environment $\mathcal{Z}$. From this point on, $\mathcal{Z}_{\mathsf{sec}}$ just executes $\mathcal{Z}$ (allowing $\mathcal{Z}$ to produce the final output).

Now we consider the ideal world game with the environment $\mathcal{Z}_{\mathsf{sec}}$ and the simulator $\mathcal{S}_{\mathsf{semi}}$. We note that, if $b = \mathsf{first}$, then this is a syntactical re-writing of $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ where the "imagined execution" produced by $\mathcal{S}$ is actually being run by the environment $\mathcal{Z}^b_{\mathsf{sec}}$. Let G be an event that this is the case (i.e. $\mathsf{first} = b$). Then conditioned on G, Game 2 matches Game 1.

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}\} \overset{c}{\approx} \{\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

Game 3 – Second Corruption adaptive Real World: We now use the semi-adaptive security of the protocol π (as simulated by $\mathcal{S}_{\mathsf{semi}}$) to switch from the ideal-world (with environment $\mathcal{Z}_{\mathsf{sec}}$ and simulator $\mathcal{S}_{\mathsf{semi}}$) to the real-world (with environment $\mathcal{Z}_{\mathsf{sec}}$ and a dummy-adversary $\mathcal{A}$). In particular we claim:

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\} \overset{c}{\approx} \{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

To see this, we note that the event G occurs with probability $1/2$ (since the view of $\mathcal{Z}$ is independent of the choice of b by $\mathcal{Z}_{\mathsf{sec}}$). Also, if G does not occur, then the output of both of games is 0. Assume that our claim does not hold, and so

$$\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid G] \geq \epsilon$$

for some ϵ which is not negligible. Then

$$\begin{aligned}
&\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} = 1] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} = 1] \\
&\quad= \Pr[G](\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid G]) \\
&\quad\quad + \Pr[\neg G](\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid \neg G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid \neg G]) \\
&\quad= \Pr[G](\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid G] - \Pr[\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} = 1 \mid G]) \\
&\quad= \epsilon/2
\end{aligned}$$

where the summand in the 3rd line is 0 since, conditioned on $\neg G$, $\mathcal{Z}_{\mathsf{sec}}$ always outputs 0. This contradicts the semi-adaptive security of π and hence out claim follows.

Game 4 – Real World: Now we notice that,

$$\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} | G\} \overset{c}{\approx} \text{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

This follows simply from the fact that, conditioned on G the left-hand side is simply a syntactic rewriting of the right-hand side, where the environment $\mathcal{Z}$ runs the protocol on behalf of P_b and passes the state of P_b to $\mathcal{Z}$.

By the hybrid argument, we therefore get

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

which completes the proof.

E.3 Proof sketch of Theorem 2.11

The simulation and the proof are similar to that of Theorem 2.10 as presented above. The main difference is that $\mathcal{S}$ needs to set-up ℓ possibilities for how to explain the communication over the ℓ-non-committing channel. Formally, the simulator $\mathcal{S}$ simulates stage (I) as in the proof of Theorem 2.10.

Then for stage (II), the simulator $\mathcal{S}$ creates ℓ copies of the simulators $\mathcal{S}^{\mathrm{prot}}_{\mathsf{semi}}$ and creates ℓ "honest-environments" ITMs $\mathcal{Z}^b_{\mathsf{sec}}(x,y)$: one (environment, simulator) pair for each choice of party b, its input $x \in X_b$ and its output $y \in Y_b$. We use the tuples (b,x,y) as indices and denote the (environment, simulator) pairs by

$$\left\{ \left(\mathcal{S}_{\mathsf{semi}}, \mathcal{Z}^b_{\mathsf{sec}}(x,y) \right) \right\}_{b\in\{0,1\}, x\in X_b, y\in Y_b}$$

Each "honest-environment" $\mathcal{Z}^b_{\mathsf{sec}}(x,y)$ runs the code of the party P_b as described by the protocol π using input x. In addition, if $\mathcal{S}^{\mathsf{sec}}$ needs to be given output on behalf of $\mathcal{F}^f_{\mathrm{SFE}}$, it is given y. To simulate the communication, $\mathcal{S}$ runs all ℓ simulations. It then initiates ℓ message-processing oracles $\mathcal{M}_i$ and randomly associates each i with an environment $\mathcal{Z}^b_{\mathsf{sec}}(x,y)$. On each tuple of messages produced by the ℓ (environment,simulator) pairs, these messages are given to the appropriate oracles and the output of the oracles is then used as side-channel information given by $\mathcal{S}$ to $\mathcal{Z}$. This is essentially a slightly more complicated version of the dummy communication phase used in the simulation for Theorem 2.10.

For stage (III), when a party P_{first} is corrupted, the simulator $\mathcal{S}$ learns the input x and (possibly) output y of the party P_{first}. It then explain the communication in stage II, by providing the internal state of the oracle corresponding to the (environment, simulator) pair for environment $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}(x,y)$ (if no input or output has been received then $\mathcal{S}$ can choose one of few consistent options for which environment to open). It passes the internal state of $\mathcal{Z}^{\mathsf{first}}_{\mathsf{sec}}(x,y)$ (i.e. its random coins and view of the protocol execution) to $\mathcal{Z}$ on behalf of P_{first}.

For stage (IV), the simulator $\mathcal{S}$ continues the simulation by using the copy of $\mathcal{S}_{\mathsf{semi}}$ chosen in the above step.

Indistinguishability of Simulation: This is similar to the proof of Theorem 2.10 and proceeds as a series of games argument.

Game 1 – The Ideal World: We define Game 1 to be $\mathrm{IDEAL}_{\mathcal{F}^f_{\mathrm{SFE}},\mathcal{S},\mathcal{Z}}$ - that is the ideal-world simulation with the environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ (as described above) interacting with the ideal functionality $\mathcal{F}_f$.

Game 2 – Second Corruption Adaptive Ideal World: This is similar to Game 2 of the proof of Theorem 2.10. The one difference is that, for stage (II), the environment $\mathcal{Z}_{\mathsf{sec}}$ now produces the side-information for $\mathcal{Z}$ by using a message-processing oracle to encrypt the actual protocol execution by P_b, and $\ell-1$ message-ignoring oracles for the rest. Again, we define the event G as in Game 2 of the proof for Theorem 2.10. It is easy to see that:

$$\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}\} \overset{c}{\approx} \{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

Indeed, the two worlds are syntactical re-writings of each other, in the sense that the left hand side has $\mathcal{Z}$ being simulated by $\mathcal{S}$ which runs a conversation $\mathcal{Z}_{\mathsf{sec}}$ and $\mathcal{S}_{\mathsf{semi}}$ while, on the right hand side, $\mathcal{Z}_{\mathsf{sec}}$ is running the whole time and is simulated by $\mathcal{S}_{\mathsf{semi}}$. The one difference is that, in Game 1, there are ℓ message-processing oracles, while in Game 2, there is only one and the rest are message-ignoring. However, by assumption, these are indistinguishable.

Game 3 – Second Corruption adaptive Real World: We now use the second-corruption adaptive security of the protocol π (as simulated by $\mathcal{S}_{\mathsf{semi}}$) to switch from the ideal-world to the real-world. In particular we get:

$$\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\mathsf{semi}},\mathcal{Z}_{\mathsf{sec}}} \mid G\} \overset{c}{\approx} \{\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} \mid G\}$$

The argument is the same as in the proof of Theorem 2.10.

Game 4 – Real World: Now we claim that,

$$\{\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\mathsf{sec}}} | G\} \overset{c}{\approx} \mathrm{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

This follows simply from the fact that, conditioned on G the left-hand side is simply a syntactic rewriting of the right-hand side, where the environment $\mathcal{Z}$ runs the protocol on behalf of P_b and passes the state of P_b to $\mathcal{Z}$.

By the hybrid argument, we therefore get

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi',\mathcal{A},\mathcal{Z}}$$

which completes the proof.

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 $\mathcal{Z}$ which follows a second-corruption adaptive adversarial strategy can distinguish the execution with the adversary $\mathcal{A}$ and OT protocol in the $\mathcal{F}_{\text{CRS}}$-hybrid world (see Figure 6), and the execution with the setup-adaptive simulator $\mathcal{S}$ and ideal functionality $\mathcal{F}_{\text{OT}}$. We further need to show the constructed simulator is input preserving as defined in Definition 2.8.

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 to set up crs_{ot} in the coins tossing stage based on the mode information. The simulation is very similar to that in [PVW08]. But now the underlying dual mode encryption satisfies an enhanced definition, and the simulator has more power to handle the second-corruption adaptive attacks from the adversaries; the simulator can use the $\mathsf{FakeEnc}$ and Recons algorithms to handle a further corruption of the sender if the receiver is initially corrupted, and use DualKG algorithm to handle a further corruption of the receiver if the sender is initially corrupted.

- **Simulating the communication with $\mathcal{Z}$.** The simulator $\mathcal{S}$ 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. Further whenever $\mathcal{A}$ corrupts a party, $\mathcal{S}$ corrupts the corresponding dummy party.
- **Trusted setup.** Note that our $\mathcal{S}$ should be a setup-adaptive simulator which includes two parts $\mathcal{S}^{\text{setup}}$ and $\mathcal{S}^{\text{prot}}$. Here we give the construction of $\mathcal{S}^{\text{setup}}$. The simulator computes the CRS in the following way: generate $(crs_{\text{com}}, \tau_{\text{com}})$ for the UC commitment, and generate $(G, crs_{\text{sys}}, \tau_{\text{sys}}) \leftarrow \mathsf{PG}_{\text{sys}}(1^\lambda)$, and further set crs_{com} and crs_{sys} as crs, and set the corresponding trapdoor τ_{com} and τ_{sys} as the trapdoor τ. When parties query $\mathcal{F}_{\text{CRS}}$, return $(\mathtt{CRS}, sid, crs)$. 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. In next several items, we construct the other part of the simulator, i.e., $\mathcal{S}^{\text{prot}}$.
- **Simulation of the initially corrupted receiver case.** Given the adversary $\mathcal{A}$ is the second-corruption adaptive adversary, here we consider the case that the receiver is corrupted initially while the sender is honest, and the sender could be further corrupted at any point in the communication stage. The simulator chooses the messy mode and simulates the sender as follows.

  First, the simulator generates $(crs_{\text{tmp}}, \tau_{\text{tmp}}) \leftarrow \mathsf{PG}_{\text{tmp}}(\text{mes}, G, crs_{\text{sys}}, \tau_{\text{sys}})$, where $(G, crs_{\text{sys}}, \tau_{\text{sys}})$ is generated in the trusted setup, and now the messy trapdoor $\tau_{\text{ot}} = (\tau_{\text{sys}}, \tau_{\text{tmp}})$ and the CRS $crs_{\text{ot}} = (crs_{\text{sys}}, crs_{\text{tmp}})$; then in the coin-tossing phase, the simulator computes s such that $crs_{\text{tmp}} = r + s$, 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 $\mathcal{F}_{\text{OT}}$ with the input bit $1-\rho$, i.e., the simulator in the name of corrupted receiver sends $(\mathtt{Receiver}, sid, 1-\rho)$ to the functionality and obtain $(\mathtt{Output}, sid, x_{1-\rho})$ 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., $(y_\rho, \omega_\rho) \leftarrow \mathsf{FakeEnc}(crs_{\text{ot}}, \tau_{\text{ot}}, pk, \rho)$. 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., $\zeta_\rho \leftarrow \mathsf{Recons}(crs_{\text{ot}}, \tau_{\text{ot}}, pk, \rho, y_\rho, \omega_\rho, x_\rho)$, and returns (ζ_0, ζ_1) as the sender's internals.
- **Simulation of the initially corrupted sender case.** Here we consider the case that the sender is corrupted initially while the receiver is honest, and the receiver could be further 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 based on $(crs_{\text{com}}, \tau_{\text{com}})$; then the simulator generates $(crs_{\text{tmp}}, \tau_{\text{tmp}}) \leftarrow \mathsf{PG}_{\text{tmp}}(\text{dec}, G, crs_{\text{sys}}, \tau_{\text{sys}})$, where $(G, crs_{\text{sys}}, \tau_{\text{sys}})$ is previously generated in the trusted setup, now the decryption trapdoor $\tau_{\text{ot}} = (\tau_{\text{sys}}, \tau_{\text{tmp}})$ and the CRS $crs_{\text{ot}} = (crs_{\text{sys}}, crs_{\text{tmp}})$; after obtaining s, the simulator computes r such that $crs_{\text{tmp}} = r + s$, 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 decrypts them into (x_0, x_1), e.g., $x_b \leftarrow \mathsf{Dec}(crs_{\mathrm{ot}}, pk, sk_b, y_b)$ for $b = 0, 1$. Then $\mathcal{S}$ sends $(\mathtt{Sender}, sid, \langle x_0, x_1 \rangle)$ to the functionality $\mathcal{F}_{\mathrm{OT}}$. Later if the receiver is corrupted the simulator, based on the learned σ, reveals sk_σ as the receiver's internals.

- **Simulation of the remaining cases.** If both parties are corrupted, the simulator just follows the internal $\mathcal{A}$'s instructions to simulate the transcripts between the two corrupted parties. If both parties are hones then the simulator runs two parties honestly based on randomly selected inputs.

We next need to argue that no PPT environment can distinguish with non-negligible probability the interaction with the protocol in the CRS hybrid world and second-corruption adaptive adversary $\mathcal{A}$, or with the functionality and simulator $\mathcal{S}$ described above.

H1 This is the CRS hybrid world.

H2 Based on H1, we change the CRS setup, and run $(G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}}) \leftarrow \mathsf{PG}_{\mathrm{sys}}(1^\lambda)$. Now trapdoor τ_{sys} is known.

H1 and H2 are indistinguishable given that the system CRS are identically distributed and τ_{sys} is not used yet.

H3 Based on H2, we change the CRS setup, generate $(crs_{\mathrm{com}}, \tau_{\mathrm{com}})$ for the UC commitment, and the trapdoor τ_{com} is known. If it is the case that the sender is honest but the receiver is initially corrupted, we do the following further modification. We generate $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(\mathrm{mes}, G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$, where $(G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$ is, as generated in H1. Next in the coin-tossing phase, compute s such that $crs_{\mathrm{tmp}} = r + s$, where r is extracted from the commitment transcripts from the corrupted receiver. Now the messy trapdoor $\tau_{\mathrm{ot}} = (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$ and the CRS $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$. If later receive a corruption command from the adversary that an honest sender is further corrupted, just reveal its input to the adversary.

H2 and H3 are indistinguishable given that the underlying commitment is UC secure and that the mode indistinguishability of the underlying enhanced dual mode encryption.

H4 Based on H3, if it is the case that the sender is honest but the receiver is initially corrupted, we do the following modification. Instead of honestly producing the ciphertexts (y_0, y_1) for both branches, we extract the messy branch number and use the faking encryption to produce the ciphertext for the messy branch, and later run the state reconstruction algorithm to compute the internals when the sender is further corrupted. For the non-messy branch, we compute the ciphertext honestly.

H3 and H4 are indistinguishable given the messy branch identification and ciphertext equivocation property of the underlying enhanced dual mode encryption.

H5 We make a further modification based on H4. Now we turn to consider the case that the sender is corrupted initially but the receiver is honest. In the coin-tossing phase, we fake the commitment transcripts from the receiver based on $(crs_{\mathrm{com}}, \tau_{\mathrm{com}})$ with the adversary. Further generate $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(\mathrm{dec}, G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$, where $(G, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$ is previously generated in H1. After obtaining s from the adversary, compute r such that $crs_{\mathrm{tmp}} = r + s$. Now the decryption trapdoor $\tau_{\mathrm{ot}} = (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$ and the CRS $crs_{\mathrm{ot}} = (crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$. Later we can equivocate the previous faked commitment transcripts into r if the receiver is further corrupted.

H4 and H5 are indistinguishable given that the underlying commitment is UC secure and that the mode indistinguishability of the underlying enhanced dual mode encryption.

H6 Based on H5, if it is the case that the sender is corrupted initially but the receiver is honest, we do the following modification. Instead of honestly producing the pk based on the receiver's input, we 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.

H5 and H6 are indistinguishable given the encryption key duality property of the underlying enhanced dual mode encryption.

H7 Based on H6, now we consider the cases that both parties are initially corrupted and that both parties are honest. For the former, just follow the adversary's instruction, and for the latter generate the transcripts honestly based on randomly selected inputs and the real inputs are not used.

H6 and H7 are indistinguishable given that the underlying dual mode encryption is secure.

H8 The ideal world running with the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}_{\mathrm{OT}}$.

In H7, the inputs for honest parties are not used. So H7 and H8 are indistinguishable. Therefore the difference between the CRS hybrid world and the ideal world are negligible.

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

Next we need to show the constructed simulator is input-preserving as defined in Definition 2.8. This can be easily verified. For the case with a corrupted sender and an honest receiver, if the corrupted sender honestly follows the protocol using input $\langle x_0, x_1 \rangle$, the input can be extracted as in the description of $\mathcal{S}$; further $\mathcal{S}$ must submit the extracted $\langle x_0, x_1 \rangle$ to the functionality, otherwise the environment can distinguish the two worlds based on the output from the honest receiver. Similarly for the case with an honest sender and a corrupted receiver, if the corrupted receiver honestly follows the protocol using input σ, the input can be extracted as in the description of $\mathcal{S}$; further $\mathcal{S}$ must submit the extracted bit σ to the functionality to learn x_σ, otherwise if $\mathcal{S}$ submit a different bit $1-\sigma$ to the functionality and learn $x_{1-\sigma}$, the simulator has no idea of x_σ, and the environment can distinguish the two worlds based on the output from the corrupted receiver.

Together we show the protocol based on Figure 6 is semi-adaptively secure to realize $\mathcal{F}_{\mathrm{OT}}$ in the CRS hybrid model. This finishes the proof.

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 $m \leftarrow \mathsf{CocDec}(sk, c)$: 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 $(crs, \tau) \leftarrow \mathsf{PG}(1^\lambda, \mu)$ for the messy mode and the decryption mode are presented as follows where $\mu \in \{\mathrm{mes}, \mathrm{dec}\}$, $crs = (\mathbb{J}_N, crs_{\mathrm{sys}}, crs_{\mathrm{tmp}})$, and $\tau = (\tau_{\mathrm{sys}}, \tau_{\mathrm{tmp}})$.

  ⋆ $(\mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}}) \leftarrow \mathsf{PG}_{\mathrm{sys}}(1^\lambda)$: Randomly select two λ-bit primes p and q and set $N \leftarrow pq$. Set $crs_{\mathrm{sys}} \leftarrow N$ and $\tau_{\mathrm{sys}} \leftarrow (p, q)$. Output $(\mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$.

  ⋆ $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(\mathrm{mes}, \mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$: Randomly select $y \xleftarrow{\$} \mathbb{J}_N \setminus \mathbb{QR}_N$. Set $crs_{\mathrm{tmp}} \leftarrow y$ and $\tau \leftarrow \emptyset$. Output $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}})$.

  ⋆ $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}}) \leftarrow \mathsf{PG}_{\mathrm{tmp}}(\mathrm{dec}, \mathbb{J}_N, crs_{\mathrm{sys}}, \tau_{\mathrm{sys}})$: Randomly select $s \xleftarrow{\$} \mathbb{Z}_N^*$ and set $y \leftarrow s^2 \bmod N$. Set $crs_{\mathrm{tmp}} \leftarrow y$ and $\tau_{\mathrm{tmp}} \leftarrow s$. Output $(crs_{\mathrm{tmp}}, \tau_{\mathrm{tmp}})$.

- The key generation algorithm $(pk, sk) \leftarrow \mathsf{KG}(crs, \sigma)$:

  Parse the crs as $(\mathbb{J}_N, N, y)$. 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 $(\mathbb{J}_N, N, y)$. Set $pk_b \leftarrow (N, pk \cdot y^b)$. Compute $(c, \zeta) \leftarrow \mathsf{CocEnc}(pk_b, m)$. Output (c, ζ).

- The decryption algorithm $m \leftarrow \mathsf{Dec}(crs, pk, sk, c)$:

  Parse the crs as $(\mathbb{J}_N, N, y)$. Compute $m \leftarrow \mathsf{CocDec}(sk, c)$. Output m.

- The messy branch identification algorithm $\rho \leftarrow \mathsf{MessyId}(crs, \tau, pk)$:

  Parse the crs as $(\mathbb{J}_N, N, y)$ 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 $\rho \leftarrow 1$; otherwise set $\rho \leftarrow 0$. Output ρ.

- The fake encryption algorithm $(c, \omega) \leftarrow \mathsf{FakeEnc}(crs, \tau, pk, \rho)$:

  Parse the crs as $(\mathbb{J}_N, N, y)$ where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p, q) where $N = pq$. Set $y' \leftarrow pk \cdot y^\rho$ and set $pk_\rho \leftarrow (N, y')$; note that $y' \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Compute $(c, \zeta_0) \leftarrow \mathsf{CocEnc}(pk_\rho, m)$, 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 $\zeta \leftarrow \mathsf{Recons}(crs, \tau, pk, \rho, c, \omega, m)$:

  Parse the crs as $(\mathbb{J}_N, N, y)$ where $y \in \mathbb{J}_N \setminus \mathbb{QR}_N$. Parse the trapdoor τ as (p, q) where $N = pq$. Set $\zeta \leftarrow \zeta_i$ where ζ_i is from ω such that $(\frac{\zeta_i}{N}) = m$. Output ζ.

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

  Parse crs as $(\mathbb{J}_N, N, y)$ and the trapdoor τ as s where $y = s^2 \bmod N$ and $N = pq$. Randomly select $r \overset{\$}{\leftarrow} \mathbb{Z}_N^*$, 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 $\{(c, \zeta) | \zeta \overset{\$}{\leftarrow} \mathbb{J}_N, c = \zeta + y/\zeta\}$; the latter distribution can be written as $\{(c, \zeta_0) | \zeta_0 \overset{\$}{\leftarrow} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, \zeta_1 = y/\zeta_0 \bmod q\}$ which can be further rewritten as $\{(c, \zeta_0) | \zeta_0 \overset{\$}{\leftarrow} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0\}$; so the two distribution are identical.

For $m = -1$, the former distribution can be written as $\{(c, \zeta) | \zeta \overset{\$}{\leftarrow} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta + y/\zeta\}$; the latter distribution can be written as $\{(c, \zeta_1) | \zeta_0 \overset{\$}{\leftarrow} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, \zeta_1 = y/\zeta_0 \bmod q\}$ which can be further rewritten as $\{(c, \zeta_1) | \zeta_0 \overset{\$}{\leftarrow} \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, (\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N})\}$; 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 $\zeta_0 \overset{\$}{\leftarrow} \mathbb{J}_N$, we can have $\zeta_1 \overset{\$}{\leftarrow} \mathbb{Z}_N^* \setminus \mathbb{J}_N$; therefore the latter distribution can be rewritten as $\{(c, \zeta_1) | \zeta_1 \overset{\$}{\leftarrow} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta_0 + y/\zeta_0 = \zeta_1 + y/\zeta_1, \zeta_1 = \zeta_0 \bmod p, (\frac{\zeta_1}{N}) = (\frac{y}{q})(\frac{\zeta_0}{N})\}$, and then $\{(c, \zeta_1) | \zeta_1 \overset{\$}{\leftarrow} \mathbb{Z}_N^* \setminus \mathbb{J}_N, c = \zeta_1 + y/\zeta_1\}$ 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. □

G DDH-based Adaptively Secure Bit and String OT

Here we give a high-level description of adaptively-secure OT constructions from the DDH-based protocol in [PVW08]. We first review the original DDH-based OT construction from PVW. Then we show how to use it to construct adaptively secure bit OT and finally string OT.

G.1 Dual Mode Encryption Based on DDH

We assume that the parties agree to work in the DDH group $\mathbb{G}$ of order p where p is chosen based on the security parameter.

- **Common Random String**: The common random string consists of four group elements $crs = (g_0, h_0, g_1, h_1)$.
  - PG("messy"). In *messy mode* the four group elements are uniformly and independently distributed. They are generated by choosing g_0, g_1 randomly in $\mathbb{G}$, (x_0, x_1) randomly in $\mathbb{Z}_p$ and setting $h_0 = g^{x_0}, h_1 = g^{x_1}$. The messy trapdoor is $t = (x_0, x_1)$. We output $\langle crs = (g_0, h_0, g_1, h_1), t\rangle$.
  - PG("decryption") In *decryption mode* the four group elements form a DDH tuple: g_0 is chosen randomly in $\mathbb{G}$ and two values x, y are chosen randomly in $\mathbb{Z}_p$. We set $g_1 = g_0^y, h_0 = g_0^x, h_1 = g_1^x = g_0^{xy}$. The decryption trapdoor is $t = y$. We output $\langle crs = (g_0, h_0, g_1, h_1), t\rangle$.

  The main difference between messy mode and decryption mode is that in decryption mode, $\mathsf{Dlog}_{g_0}(h_0) = \mathsf{Dlog}_{g_1}(h_1)$ while in messy mode this happens with negligible probability.

- $\mathsf{KG}(\sigma)$: Given a bit $\sigma \in \{0,1\}$, choose $r \leftarrow \mathbb{Z}_p$. Let $g = g_\sigma^r, h = h_\sigma^r$. Set $pk = (g, h)$, $sk = r$. Output (pk, sk, σ).

- $\mathsf{Enc}(pk, m, b)$: Given a public key $pk = (g, h)$, a message m and a bit b, choose $s, t \leftarrow \mathbb{Z}_p$ and set $u := g_b^s h_b^t$ $v := g^s h^t$. Set the ciphertext $c = (u, v \cdot m)$. Output (c, b).

- $\mathsf{Dec}(sk, b, \sigma, c)$: Given a secret key $sk = r$ with label σ and a ciphertext $c = (c_0, c_1)$ with label $b = \sigma$, output $m' = c_0^r / c_1$.

- $\mathsf{MessyId}(pk, t)$: Given the messy trapdoor $t = (x_0, x_1)$ and a public key $pk = (g, h)$ test for $h = g_0^x$. If the test passes, label the key as a left-key by outputting $e = 0$. Otherwise label it as a right key by outputting $e = 1$.

- $\mathsf{DualKG}(t)$: Given the decryption trapdoor $t = y$, we can compute a key pair (pk, sk) which can be used to decrypt *both* left and right decryptions. To do so compute $(g, h) = (g_0^{r_0}, h_0^{r_0})$, where r_0 is chosen randomly, and then compute $r_1 = r/y$ so that $(g, h) = (g_1^{r_1}, h_1^{r_1})$. Output $(pk, sk = (r_0, r_1))$. Later, we can call the Dec procedure with secret keys r_0 or r_1 to decrypt both left and right ciphertexts.

G.2 Bit OT

There is one main problem in the above scheme when it comes to adaptive security. When the key and the encryption type do not match in messy mode (i.e. a left encryption under a right key) the encryption is statistically hiding. However, for adaptive security, the simulator also needs to be able to generate a valid looking ciphertext of this form and later be able to explain it as a valid encryption of any specified message. Unfortunately, in the current scheme, this might not be possible since the encryption can be binding even for a simulator who knows the messy trapdoor.

To see this in detail, recall that the receiver can choose an arbitrary public key g, h. If $h \neq g^{x_0}$, then left encryption is statistically hiding since $u = g_0^s h_0^t = g_0^{s+tx_0}, v = g^s h^t = g_0^{r(s+tx_1')}$ for some $r, x_1' \neq x_0$ and hence u, v are statistically independent. However, for any left ciphertext (c_0, c_1), if the simulator can explain the ciphertext as an encryption of either one of two different messages $m \neq m'$, then the simulator can compute s, t, s', t' such that $g^s h^t m = c_1 = g^{s'} h^{t'} m'$. This presents two problems. Firstly, the simulator must know $\mathsf{Dlog}_g(h)$ but g, h can be chosen arbitrarily by the receiver. Secondly, the simulator must know $\mathsf{Dlog}_g(m)$ but m is also chosen arbitrarily without input from the simulator.

The second problem is the easier of the two to solve - we simply shrink the message space of the encryption scheme to 1 bit: to encrypt a bit $e \in \{0,1\}$ we simply use the original scheme on the message $m = g^e \in \{1, g\}$. This way, the simulator knows $\mathsf{Dlog}_g(m)$ for both eligible messages.

The first problem is a little trickier, but we notice that the honest receiver always chooses g, h such that $\mathsf{Dlog}_g(h) \in \{x_0, x_1\}$. Somehow we need to disallow the cheating receiver from choosing arbitrary g, h. To do so, we want the receiver to prove that either $\mathsf{Dlog}_g(h) = \mathsf{Dlog}_{g_0}(h_0)$ or $\mathsf{Dlog}_g(h) = \mathsf{Dlog}_{g_1}(h_1)$. Actually, we can make the receiver prove an equivalent statement that either $\mathsf{Dlog}_{g_0}(g) = \mathsf{Dlog}_{h_0}(h)$ or $\mathsf{Dlog}_{g_1}(g) = \mathsf{Dlog}_{h_1}(h)$. We take advantage of the fact that there are efficient Σ protocols for the equality of discrete logs. Actually we need the Σ protocol to be *non-erasure* in the sense that if the receiver, after generating the transcript with the sender, is corrupted, the simulator can "explain" such Σ protocol transcript. Further the Σ protocol is required to defend against a dishonest verifier; we can use a trick by Damgård [Dam00] to transform a Σ protocol against honest verifier into one against dishonest verifier by using an equivocal commitment to "hide" the first move of the Σ protocol until receiving the challenge from the verifier. Please refer to [Nie03, Section 2.9] and [Nie05] for more details. Let (a, c, z) be the three moves of the non-erasure Σ-protocol against a dishonest verifier. The protocol therefore becomes:

- Let $crs_{\text{ot}} = (g_0, g_1, h_0, h_1)$.
- The receiver chooses a key $pk = (g, h), sk$ using the key generation algorithm on his bit σ. In addition the receiver computes the first message a of a Σ-protocol. Here the relation associated with the Σ-protocol is defined as $\mathtt{R} = \{(pk, crs_{\text{ot}}), r | (g = g_0^r \wedge h = h_0^r) \vee (g = g_1^r \wedge h = h_1^r)\}$.
- The sender sends a challenge c.
- The receiver sends a response z.
- The sender verifies that (a, c, z) is accepting and, if so, computes a left encryption of $m_0 = g^{e_0}$ for the left message $e_0 \in \{0,1\}$ and a right encryption of $m_1 = g^{e_1}$ for the right message $e_0 \in \{0,1\}$ under pk.

The above yields a semi-adaptively secure bit-OT protocol. We now apply our compiler from Section 2.4, and use somewhat non-committing encryption to protect the protocol transcripts to obtain a very efficient adaptively secure bit OT based on the DDH assumption. We further remark that short (logarithmic size) string OT can be similarly obtained.

G.3 String OT

The above approach for semi-adaptively secure bit OT based on DDH can be *efficiently* extended to obtain semi-adaptively secure string OT, for larger strings of n bits. There are two issues that we must face in order to convert the bit-OT scheme into a string-OT scheme. Firstly, we cannot simply increase the message space to all of $\mathbb{Z}_p$ since that would not allow for efficient equivocation. Secondly, we cannot use our standard somewhat non-committing compilation since such a compiler is only efficient for small domains/ranges. Here we take a detour; we simply "bundle" n copies of bit OT into a string OT for n bits. This can be done as follows. First the receiver computes an equivocal commitment of his bit σ. Then, the sender and the receiver run n copies of bit-OTs in parallel, and the Σ-protocol guarantees not only that each "public key" pk chosen by the receiver is well-formed, but also that all public keys are based on same bit σ, which matches the commitment. Using Pedersen commitments, we have efficient Σ-protocols for this language.

This already gets us an adaptive OT protocol assuming secure channels. However, we now show how to efficiently use somewhat non-committing encryption to improve computational efficiency. The idea is that each of the seperate bit-OT protocols is executed over its own "ℓ-equivocal channel" (essentially a fresh independent channel for somewhat NCE). It is easy to show that this still allows the simulator to simulate the "initial corruption". Moreover, we notice that it is very efficient to set up n separate ℓ-NCE channels using (strict) $O(n)$ public key operations and strict constant number of rounds.[5] This results in the parameters mentioned in Theorem 3.4.

[5]Essentially, we need to use fully non-committing encryption to send n indices in the range $[1, \ell]$. We can do so with the above efficiency assuming $n = \Omega(\lambda)$. Recall that previously, for NCE of a single bit, we did not wish to sacrifice $O(\lambda)$ operations and thus were forced to use *expected* constant number of rounds/operations.

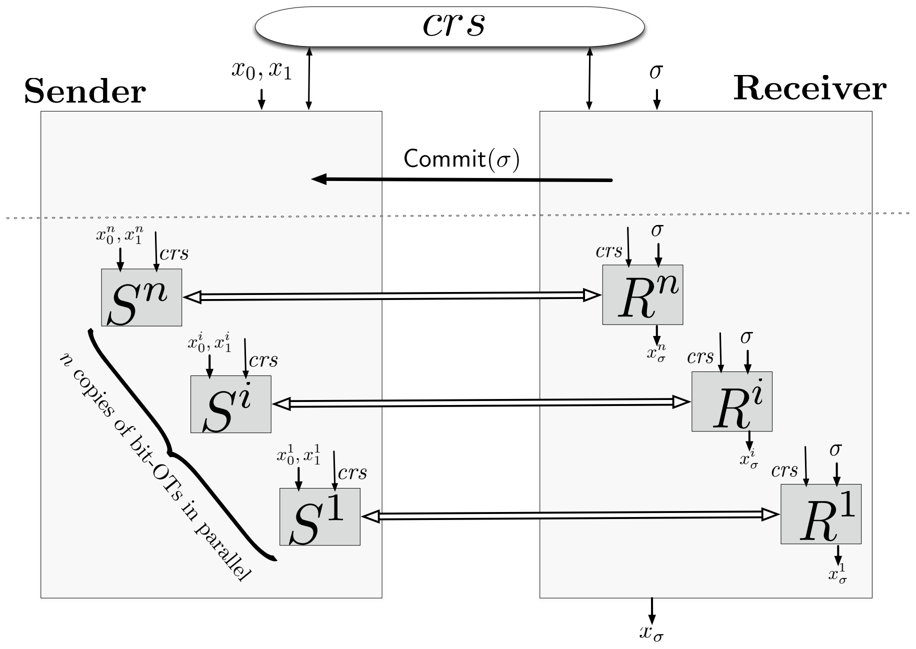


Figure 8: DDH-based fully adaptively-secure String-OT protocol. The protocol consists of two stages. In the first stage, the receiver sends an equivocal commitment on its input σ; for the purpose of giving a concrete example, we here instantiate the equivocal commitment with Peterson commitment $c = \mathsf{Commit}(\sigma) = g_\Delta^\sigma h_\Delta^\gamma$. In the second stage, the sender and the receiver run n copies of fully adaptively-secure bit-OTs explored in Section G.2 but with the relation updated in the Σ protocol, i.e., in i-th bit-OT, $\boxed{S^i} \Longleftrightarrow \boxed{R^i}$, the relation is updated into $\mathtt{R}^i = \{(c, pk^i, crs^i_{\text{ot}}), (r^i, \gamma) | (g^i = (g^i_0)^{(r^i)} \wedge h^i = (h^i_0)^{(r^i)} \wedge c = h^\gamma_\Delta) \vee (g^i = (g^i_1)^{(r^i)} \wedge h^i = (h^i_1)^{(r^i)} \wedge c = g_\Delta h^\gamma_\Delta)\}$ where $crs^i_{\text{ot}} = \langle g^i_0, h^i_0, g^i_1, h^i_1, \mathbb{G}\rangle$ is obtained by S^i and R^i after coin tossing phase, and pk^i is in the first move in transferring phase. Further $\mathbb{G} = (G, p, g_*)$ is a cyclic group with prime order p and generator g_*, and $g^i_0, h^i_0, g^i_1, h^i_1, g_\Delta, h_\Delta \in G$.