Universally Composable Committed Oblivious Transfer and Multi-Party Computation Assuming Only Basic Black-Box Primitives

Gregory Estren

School of Computer Science
McGill University, Montreal
August, 2004

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Science

Abstract

Committed oblivious transfer (COT) is a straightforward combination of bit commitment and oblivious transfer that is powerful enough to achieve general multi-party computation with no additional assumptions. We show how to securely implement COT and multi-party computation in the universally composable (UC) framework. Our protocol only requires access to underlying UC commitment, UC oblivious transfer, and UC authentication functionalities. It is the first such protocol that achieves this rigorous notion of security with no explicit computational assumptions.

Résumé

Le transfert inconscient mis en gage, ou *commited oblivious transfer*, (COT) est le fruit d'une combinaison directe d'une mise en gage, ou *bit commitment*, et d'un transfert inconscient, ou *oblivious transfer*, qui est suffisamment puissante afin de faire des calculs à plusieurs partis sans autres hypothèses additionelles. Nous démontrons comment implanter de façon sûr le COT et les calculs à plusieurs partis dans le cadre de la composition universelle (UC). Notre protocole ne nécessite que l'accès aux fonctionalités de mise en gage UC, de transfert inconscient UC, et d'authentification UC fondamentales. C'est le premier protocole à satisfaire cette notion rigoureuse de sécurité sans l'usage d'hypothèses calculatoires explicites.

Acknowledgements

Thank you to Claude Crépeau for providing the idea behind this thesis, maintaining endless patience during my efforts to understand the problem, and engaging in numerous invaluable discussions that were instrumental in bringing everything to fruition.

Thank you to Abdul Ahsan, Martin Courchesne, Simon Pierre Desrosiers, Kirill Morozov, and Raymond Putra for their gracious help on research-related issues.

Finally, thank you to Geneviève Arboit, Rebecca Barnes, Matthew Demanett, Geralyn Estren, Mathieu Petitpas, George Savvides, and Eugenia Van Bremen for their positive company and general support. Also thank you to Michael Spiegel for teaching me everything I know.

Contents

Chapter 1

Introduction

1.1 Secure Multi-Party Computation

Say that two coworkers wish to compare their salaries. Neither is willing to reveal his salary to the other, but both want to know who makes more money.[1] The process of achieving this goal is known as a *secure multi-party computation.* More generally, a multi-party computation consists of n mutually distrustful parties conducting a protocol that maps their private inputs into some combined output. (when $n = 2$, this is known as a *two-party computation*). A secure computation must intuitively satisfy the notions of *privacy* and *correctness.* Privacy guarantees that the protocol reveals no information about the inputs beyond that deducible from the output. Correctness guarantees that parties can't cheat such that the protocol produces untrue results. Additional security concerns include *fairness* (either all parties receive output or no party receives output) and *input independence* (no party can correlate its input with another party's unknown input), among others.

[1] This example is a variation of Yao's *millionaires problem* from [50]. See [37] for more examples.

We represent attacks against a protocol by dividing participants into honest parties and corrupted parties. Honest parties always follow the protocol exactly as specified. Corrupted parties instead follow the directions of an independent party known as the *adversary*. The adversary has full access to a corrupted party's private data and may arbitrarily coordinate actions among multiple corrupted parties. Adversaries may be *semi-honest* (*passive*) or *malicious* (*active*). Semi-honest adversaries follow the protocol correctly but attempt to learn more information than intended. Malicious adversaries deviate from the protocol in arbitrary ways. Adversaries may also be *static* or *dynamic*. Static adversaries control a fixed set of corrupted parties throughout the protocol. Dynamic adversaries can corrupt new parties at any point during the computation.

As in many areas of cryptography, finding adequate formal definitions for secure multi-party computation has been a delicate and difficult process. Yao first formulated the problem in [50] in the context of secure function evaluation, where computations are functions of fixed inputs. He later produced the first general protocol for secure two-party function evaluation in the computational setting ([51]). Goldreich, Micali, and Wigderson introduced the first general protocol for multi-party computation in the computational setting, assuming less than half of the parties are corrupted ([35]). Their protocol supports not only function evaluation but also "mental game" computations that can be reactive in nature. Further research produced improvements in efficiency ([28], [4]), majority-corruption scenarios ([3], [14]), and information-theoretic security ([16], [5], [15]).

The security definitions in [51] and [35] eventually proved inadequate. The main problem was that they identified specific security requirements (e.g. privacy and correctness) and based security around fulfilling these requirements. But these requirements are by no means exhaustive. For example, the definition in [51] doesn't account for input independence.[2] While we can add input independence as a new requirement, there may still be other requirements that haven't been identified.

This led to a series of new definitions that attempted to resolve these inadequacies. Most notable are the definitions of [38], [44], and [1]. They introduce the crucial paradigm of evaluating a protocol's security by comparing it to an *ideal protocol* that has access to a trusted party. All parties in an ideal computation forward their inputs directly to the trusted party, which performs the required computation and delivers the correct output back to each party. The trusted party never reveals any information beyond the required outputs and cannot be corrupted by the adversary. An ideal protocol is therefore trivially secure. A real protocol is secure if it is "equivalent" to an ideal protocol under adversarial attack. The exact nature of this equivalence varies between definitions.

1.2 Protocol Composition

While [38], [44], and [1] provide effective general solutions for multi-party computation, they only consider *stand-alone security*, where a protocol runs exactly once and in isolation. In contrast, realistic scenarios often involve a protocol running multiple times in the presence of other parties and com-

[2] If an honest party in a two-party computation has private input x, [44] observes that Yao's definition allows the other party to force the computation's output to $f(x, x)$. This is possible even without knowing x.

putations with unknown behavior.[3] Stand-alone security does not work for such scenarios. This is an important concern. A security model that fails for realistic scenarios offers very little true security. Consequently, there has been significant research over the last decade in adapting stand-alone security models to multi-execution settings.

When multiple protocols run together, this is known as *composition*. If a protocol composes with itself, this is known as *self composition*. If a protocol composes with arbitrary protocols, this is known as *general composition*. Composition generally occurs in one of three ways:

- *sequential composition* - No two protocol executions occur at the same time. In other words, for any two executions A and B, either A finishes before B starts or B finishes before A starts.
- *parallel composition* - All executions start at the same time and progress at the same rate (according to a synchronized round-based schedule).
- *concurrent composition* - Protocol execution is scheduled arbitrarily. This is the most general form of composition (it includes sequential and parallel composition as special cases).

The most common cryptographic protocols have been explicitly extended to support composed settings. For encryption, Goldwasser and Micali's stand-alone definition of semantic security ([39]) has been extended to handle chosen-ciphertext attacks, where the adversary has concurrent access to decryption functionality ([45], [46]). For zero-knowledge, Goldwasser, Micali, and Rackoff's stand-alone definition ([40]) has been extended into sequential ([36], [34]),

[3] Imagine two people flipping a virtual coin over the internet. They may want to flip many coins instead of just one. Meanwhile, they may be checking email, downloading files, etc., while other unknown parties are trying to break into their systems.

parallel ([34], [33]), and concurrent ([26], [47], [21], [42], [32], etc.) settings. For oblivious transfer, original definitions have been extended into the concurrent setting ([29], [2]).

In [25], Dolev, Dwork, and Naor identified a specific attack that relies on composition and introduced the concept of *non-malleability* as a means to block it. Specifically, they highlighted the concern that an adversary may feed one protocol instance's output into another's input to gain some advantage. This is commonly known as a "man in the middle" attack. [25] defines what it means for a protocol to maintain security against this attack and develops non-malleable solutions for encryption, commitment, and zero-knowledge. This spawned significant subsequent research in non-malleable protocols ([19], [48], [27], [20], etc.).

For general multi-party computation, both the [44] and [1] security definitions maintain security under limited forms of composition. Canetti updated these models in [9] to form a simpler and less restrictive definition that provides rigorous security under general non-concurrent composition.

Unfortunately, all of these approaches have severe limitations. Concurrent definitions of specific protocols usually only work in self-composed settings, i.e. they fail in the presence of arbitrary protocols. These definitions also have limited applicability, as each protocol requires its own distinct definition. Non-malleability offers a single security definition for all protocols, but it only protects against one type of attack. While we can always identify new attacks and incorporate them into updated security definitions, this results in continuously changing definitions that are inherently vulnerable to unknown attacks. Finally, while general multi-party computation definitions can escape these problems, none of the above definitions work for concurrent settings.

What we really want are simple and general security definitions that maintain security under any form of composition. In other words, we want security definitions that "just work", regardless of how they're used.

1.3 Universally Composable Security

In [7], Canetti presented a framework that finally offers rigorous security for any protocol under any type of composition. This is known as the *universally composable framework*. It consists of a security definition and composition theorem, informally described as follows:

As in the definitions of [44], [1], and [38], security for a protocol comes from showing equivalence to an idealized version of the protocol that has access to a trusted third party. The idealized protocol runs in the *ideal model* of computation, with the trusted party is known as an *ideal functionality*. The regular protocol runs in the *real model* of computation. Both models contain two special parties known as the *adversary* and the *environment*. The adversary delivers all messages between parties and corrupts parties at will. The environment provides all protocol inputs and reads all outputs, but cannot see protocol messages. The adversary and environment may privately communicate with each other in arbitrary ways. Roughly speaking, a protocol is secure if no environment can ever tell if it is using the real protocol in the real model or the idealized protocol in the ideal model, regardless of the adversary's behavior. Protocols that meet this requirement are known as *universally composable* (*UC*) and are said to *UC realize* an ideal functionality.

What distinguishes UC security from other security definitions is that a universally composable protocol remains secure under general concurrent composition. This means security is automatically preserved under any imaginable usage context. The *universal composition theorem* provides this guarantee. The theorem also permits a very convenient "plug and play" form of protocol design that allows one to design a secure protocol that uses an ideal functionality and then replace this functionality with an equivalent UC sub-protocol with no loss of security.

Large classes of functionality are known to have universally composable implementations. This includes authentication ([7]), secure message transmission ([7]), commitment ([10], [23]), zero-knowledge ([10]), and general multi-party computation ([12], [30], [24]). One caveat of these results is that most functionalities cannot be realized without some kind of trusted party access ([11], [10], [7]). A common reference string is sufficient for any purpose ([12]).

1.4 Our Results

This thesis presents a UC general multi-party computation protocol that is secure against malicious, adaptive adversaries for any number of corrupted parties and requires no explicit computational assumptions. It is based on a primitive known as *committed oblivious transfer* (COT), which is a variation of oblivious transfer that uses commitments for inputs and outputs.[4] Crépeau, van de Graaf, and Tapp describe in [18] an efficient COT protocol and show how it can be used to achieve multi-party computation. Our contribution essentially consists of translating their results into the UC framework. We define a UC COT protocol that assumes only the existence of UC bit commitment, UC oblivious transfer, and UC authentication. We then plug this into a multi-

[4]This was first defined in [17] under the name "verifiable oblivious transfer".

party circuit evaluation protocol that requires no additional assumptions. For n parties evaluating a q-gate circuit with security parameter m, this achieves a running time of $O(n^2qm^2c + nqmt)$, where c is the running time of a bit commitment and t is the running time of an oblivious transfer.

[12] introduced the first UC multi-party computation protocol that works with corrupted majorities. Their protocol follows the "GMW compiler" paradigm from [35]. That is, they first develop a protocol secure against semi-honest adversaries and then "compile" it into a protocol secure against malicious adversaries by having all parties show that their messages are correct through the use of zero-knowledge proofs. While this approach works under general assumptions (such as trapdoor permutations), the compiler requires full access to the "source code" of the semi-honest protocol. In particular, the compiler cannot handle protocols based on noisy channels, quantum channels, or black-box primitives where no source code is available.[5] In contrast, our protocol uses its required bit commitment, oblivious transfer, and authentication primitives in a purely black-box manner. It makes no assumptions whatsoever about how they work. This allows us to support all computational settings, restricted only by how we choose to implement these primitives.

[23] and [30] also present protocols for UC multi-party computation, but they focus on efficiency rather than generality. The protocol in [23] runs in $O(nmq)$ time (for n parties, security parameter m, and a q-gate circuit), but relies on specific number-theoretic assumptions and cannot handle corrupted majorities. The protocol in [30] runs in just $O(q)$ time, but also relies on specific number-theoretic assumptions and only works in the *erasing model*, where honest parties are trusted to reliably erase their private data when

[5] More generally, the compiler cannot handle protocols that cannot be verified through zero-knowledge proofs.

they no longer need it. To our knowledge, our protocol has the most general assumptions of any known protocol that offers comparable security.

We organize the remaining chapters as follows: Chapters 2 and 3 provide detailed descriptions of the UC framework and COT, respectively. Chapter 4 provides a UC implementation of "bit commitment with XOR", a useful tool that allows us to perform limited zero-knowledge proofs on committed bits. Chapter 5 defines the COT protocol from [18] and proves its security in the UC framework. Chapter 6 shows how to use COT to achieve general two-party and multi-party computation. Chapter 7 offers final analysis and conclusions.

Chapter 2

Universally Composable Security

This chapter describes universally composable (UC) security in detail. We first review relevant fundamental concepts of computation, then provide a formal description of the UC framework, and finally provide UC definitions for cryptographic primitives important to our protocol.

2.1 Fundamentals

2.1.1 Multi-Party Computation

A *multi-party computation* consists of n parties $P_1, ..., P_n$ producing a sequence of private outputs from a sequence of private inputs.[1] When this process can be represented as a function $f(x_1, ..., x_n, r) = (y_1, ..., y_n)$ for input sequence $(x_1, ...x_n)$, output sequence $(y_1, ..., y_n)$, and random data r, this is known as *function evaluation*. The salary comparison example from the introduction

[1]If $n = 2$, this is known as a *two-party computation*.

is an example of function evaluation. This scenario can be modeled by two parties P_1, P_2 with inputs $x_1 = P_1$*'s salary*, $x_2 = P_2$*'s salary* and function $f(x_1, x_2) = (1, 1)$ if $x_1 > x_2$, $(2, 2)$ if $x_2 > x_1$, and $(0, 0)$ if $x_1 = x_2$.[2] Function evaluation is a special case of multi-party computation that determines outputs from a fixed set of inputs. More general computations can be *reactive*, where parties receive new inputs throughout the computation.

A multi-party computation is *secure* if dishonest parties cannot interfere with it in any way except by choosing their inputs arbitrarily. We formalize this notion when we define UC security.

2.1.2 Indistinguishability

For $w \in \{0,1\}^*$, let X_w be a probability distribution ranging over strings of length polynomial in $|w|$. A *probability ensemble* $X = \{X_w\}_{w\in\{0,1\}^*}$ is an infinite set of probability distributions ranging over all values of w.

Intuitively speaking, two ensembles are (computationally) indistinguishable if no polynomial-time algorithm, given a sample, can tell which ensemble that sample comes from. For example, we can view a probabilistic program's output as a probability distribution on its input. Two programs produce indistinguishable output if no polynomial-time adversary, given some output, can tell which program it came from. More formally:

Definition 2.1.1 *Two ensembles* $X = \{X_w\}_{w\in\{0,1\}^*}$ *and* $Y = \{Y_w\}_{w\in\{0,1\}^*}$ *are* computationally indistinguishable *if for every probabilistic polynomial-time algorithm* D *and every polynomial* $p(\cdot)$*, there exists* $w_0 \in \{0,1\}^*$ *such that for all* w *where* $|w| > |w_0|$*,*

[2] Although each output is private, in this case both parties receive the same output, so it essentially becomes public.

$$|Pr[D(X_w, w) = 1] - Pr[D(Y_w, w) = 1]| < \frac{1}{p(|w|)}$$

If X and Y are indistinguishable, we write $X \approx Y$.

In the UC framework, a probability distribution takes the form $X(m, a)$, for $m \in \mathbb{N}$ and $a \in \{0, 1\}^*$. This notation is used to describe the output probability of an algorithm running with security parameter m and input a. Furthermore, the UC framework only considers *binary ensembles*, where probability distributions range over $\{0, 1\}$. This allows an alternate definition for indistinguishability:

Definition 2.1.2 *Two binary ensembles $X = \{X(m, a)\}_{m \in \mathbb{N}, a \in \{0,1\}^*}$ and $Y = \{Y(m, a)\}_{m \in \mathbb{N}, a \in \{0,1\}^*}$ are computationally indistinguishable if for every polynomial $p(\cdot)$, there exists $m_0 \in \mathbb{N}$ such that for all $m > m_0$ and for all a,*

$$|Pr[X(m, a) = 1] - Pr[Y(m, a) = 1]| < \frac{1}{p(m)}$$

We sometimes refer to computationally indistinguishable ensembles simply as *indistinguishable*.

2.1.3 Interactive Turing Machines

The UC framework models each party as an *interactive Turing machine*, which is a Turing machine that can communicate with other machines. More formally:

Definition 2.1.3 *([7], following the definition of [31]) An interactive Turing machine (ITM) M is a Turing machine with the following tapes:*

1. *a* read-only input tape *that holds M's private input.*

2. *a* read-only random tape *that holds M's random coin input.*

3. *a* read-only security parameter tape *that specifies the protocol's security parameter.*

4. *a* write-only output tape *that holds M's private output.*

5. *a* read-and-write work tape *used for private, internal computations.*

6. *a* read-only identity tape*, that specifies M's* identity*. Every participant in a multi-party protocol has a unique identity.*

7. *a* read-and-write one-bit activation tape*. When this tape is set to 1, we say that M is* activated*. Upon activation, M follows its program and eventually enters either a* waiting state *or a* halt state*. When M enters a waiting state, it sets this tape to 0 and remains idle (does not change its state, tape contents, or head positions) until its next activation. When M enters a halt state, it sets this tape to 0 and remains idle through all future activations (overriding the original activation rule).*

8. *a* read-only incoming communication tape *that holds* incoming messages *from other parties. Each message consists of a* sender field*, which contains the sender party's identity, and a* contents field*, which contains arbitrary data.*

9. *a* write-only outgoing communication tape *that holds* outgoing messages *destined for other parties. Each message consists of a* recipient field*, which contains the identity of the intended recipient, and a* contents field*, which contains arbitrary data.*

A multi-party computation informally consists of ITMs receiving initial inputs, undergoing computation through an ordered series of activations and message communication, and producing final outputs. The exact details of how this works vary among different setup assumptions. For example, in the two-party model of [31] parties communicate *directly.* That is, one party's outgoing communication tape is the same as the other party's incoming communication tape. This guarantees instant and reliable message delivery. In contrast, the UC model of computation requires an untrusted third party to deliver messages. When a party writes a message onto its outgoing communication tape, the third party reads this message and copies it onto the recipient's incoming communication tape at its leisure. This makes communication less reliable. We formalize this process in the next section.

2.2 Universally Composable Framework

2.2.1 Summary and Intuition

As described in the introduction, the universally composable framework consists of a security definition and a composition theorem. The security definition asserts what it means for a protocol to be universally composable. The composition theorem guarantees that universally composable protocols remain secure under general composition (the composition theorem is what distinguishes UC security from other security definitions). Each party is modeled by an interactive Turing machine.

In any multi-party computation, the best security we can hope for is that all parties forward their inputs to a mutually trusted and incorruptible party F, which internally performs the entire computation and forwards appropriate outputs back to each party. This is known as the *ideal model* of computation. In this model, a party's only function is to forward its inputs directly to F and forward F's responses directly to its output. F is known as an *ideal functionality*

Unfortunately, realistic protocols generally don't have access to trusted parties. Instead, parties must find some safe and reliable way to interact with each other in spite of their mistrust. This is known as the *real model* of computation. In this model, the burden of computation falls on the parties themselves rather than an ideal functionality.

Both models contain a party known as the *adversary* (the ideal model adversary is known as S, whereas the real model adversary is known as A). The adversary can corrupt parties and read communication messages.[3] The adversary also controls message "delivery". That is, when P_i wants to send a message to P_j, P_i writes this message on its outgoing communication tape. It then becomes the adversary's responsibility to copy this message onto P_j's incoming communication tape. This models the intuition that in realistic networks (e.g. the internet), communication is not necessarily reliable. Finally, the adversary cannot read messages to and from the ideal functionality in the ideal model.

Both models also contain a party known as the *environment*.[4] The environment "runs" the protocol by providing all parties with inputs and reading their outputs. It cannot, however, read communication messages. This models the intuition that one can use a protocol without knowing how it internally works.

[3] The adversary cannot, however, modify message contents or forge messages. This means we assume ideally authenticated communication. We bring this issue up again in chapter 7.

[4] As the name implies, the environment represents the broader context in which the protocol is run. Any protocol that doesn't run in isolation must be a component of a broader system.

The environment also communicates arbitrarily with the adversary. This models the intuition that the broader world may be malicious in unknown ways.[5]

Say we want to evaluate the function f using protocol ρ. Let F be the ideal functionality that computes f on the protocol inputs in the ideal model. We say that ρ *UC realizes* F if for every real adversary A, there exists an ideal adversary S such that no environment can distinguish between ρ running with A in the real model and F running with S in the ideal model. In other words, our protocol is secure if no environment can ever distinguish between the real world and the ideal world. This means that no matter what the context, running the real protocol is equivalent to running the ideal protocol, which is secure by definition. Protocols that satisfy this requirement are known as *universally composable* (or *UC secure*).

The *universal composition theorem* guarantees that secure protocols remain secure under general composition. Informally, the theorem states the following:

> For an ideal functionality F, let π^F be any protocol that uses F. Let π^ρ be the same protocol with all calls to F replaced by equivalent calls to a protocol ρ. If ρ UC realizes F, then no environment can distinguish between π^F and π^ρ.

An important corollary of the theorem permits a "plug and play" model of protocol design. It informally states:

> For ideal functionalities F and G, define π^F, ρ, and π^ρ as above. If π^F UC realizes G and ρ UC realizes F, then π^ρ UC realizes G.

[5]Note that the adversary has access to protocol messages but not input and output, while the environment has access to input and output but not protocol messages. Since the adversary and environment can arbitrarily communicate with each other, this seems to imply no need for defining them separately. However, assuming full knowledge sharing restricts the generality of the security model. We can achieve useful results by having the environment withhold information from the adversary.

With this result, we can build a complicated protocol as follows: if the protocol requires a sub-protocol (say a zero-knowledge proof), design it such that it uses ideal zero-knowledge functionality and prove that it is secure. Then take any universally composable zero-knowledge protocol ρ and replace the ideal functionality with ρ. The modified protocol remains secure.

We now formally define the UC framework (as defined in [12]).

2.2.2 The Real Computation Model

A real model n-party computation of protocol π consists of real parties $P_1, ..., P_n$, real world adversary A, and environment Z (all modeled as ITMs). All parties start with infinitely long random data (chosen from a uniform distribution) written onto their random tapes. Z additionally starts with some value z written onto its input tape (z is assumed to contain the local inputs for $P_1, ..., P_n$). All parties have security parameter m. Protocol execution begins with Z activated and all other parties in the waiting state. Protocol execution ends when Z halts. The protocol output is Z's output, assumed to be a single bit.[6] Parties behave as follows:

1. **Z:** Upon activation, Z may perform internal computations, process its own tapes, read the output tapes of $P_1, ..., P_n$ and A, and write to the input tape of at most one party (one of $P_1, ..., P_n$ or A). If Z writes to a party's input tape, Z enters the waiting state and activates that party. Otherwise, Z halts.

2. **Uncorrupted P_i:** Upon activation, an uncorrupted P_i may perform internal computations and process its own tapes. It ends its activation

[6]Note that assuming single-bit output does not reduce the model's generality. This is because Z's purpose is to serve as a distinguisher between the real and ideal models.

by entering the waiting state or halt state and activating Z.

3. **A:** Upon activation, A may perform internal computations, process its own tapes, read the outgoing communication tapes of $P_1, ..., P_n$, deliver a message between parties, and corrupt some $P \in \{P_1, ..., P_n\}$. A delivers a message m from P_i to P_j by copying m from P_i's outgoing communication tape to P_j's incoming communication tape (note that A cannot modify m). When A corrupts P, A learns P's entire state history and writes a message to its output tape notifying Z of the corruption. P can no longer be activated. When A halts or enters its waiting state, if A has delivered a message to $P_j \in \{P_1, ..., P_n\}$ then A activates P_j. Otherwise A activates Z.

2.2.3 The Ideal Computation Model

The ideal model is similar to the real model except it defines an additional party known as the *ideal functionality*. The ideal functionality is a trusted, incorruptible party that takes the other parties' inputs, performs computations, and responds with appropriate outputs. Every protocol has a unique ideal functionality (since different protocols have different computation and security requirements).

An ideal model n-party computation consists of "dummy" parties $P_1, ..., P_n$[7], ideal functionality F, ideal world adversary S, and environment Z (all modeled as ITMs). As in the real model, all parties start with infinitely long random data (chosen from a uniform distribution) written onto their random tapes. Z additionally starts with some value z written onto its input tape (z is assumed

[7]We call them "dummy" parties because they simply relay messages between the environment and the ideal functionality.

to contain the local inputs for $P_1, ..., P_n$). All parties have security parameter m. Protocol execution begins with Z activated and all other parties in the waiting state. Protocol execution ends when Z halts. The protocol output is Z's output, assumed to be a single bit. Parties behave as follows:

1. **Z:** Upon activation, Z may perform internal computations, process its own tapes, read the output tapes of $P_1, ..., P_n$ and S, and write to the input tape of at most one party (one of $P_1, ..., P_n$ or S). If Z writes to a party's input tape, Z enters the waiting state and activates that party. Otherwise, Z halts. *Note that Z has no access to F.*

2. **Uncorrupted P_i:** If P_i is activated due to new input from Z, P_i copies this input to its outgoing communication tape (destined for F), enters the waiting state, and activates Z.[8] If P_i is activated due to a new message from F, P_i copies this message onto its output tape, enters the waiting state, and activates Z. *Note that P_i cannot perform its own computations or send messages to anyone but F.*

3. **F:** Upon activation, F may read its incoming communication tape, perform internal computations, and write messages on its outgoing communication tape destined for $P_1, ..., P_n$ or S. F then enters the waiting state and activates Z.

4. **S:** Upon activation, S may perform internal computations, process its own tapes, deliver a message from some P_i to F or from F to P_i, and corrupt some $P_i \in \{P_1, ..., P_n\}$. S delivers a message from P_i to F by copying the message from P_i's outgoing communication tape onto F's

[8] This contrasts with the definition of [7], where parties write directly onto F's incoming communication tape (no message delivery is required). We follow the approach of [12].

incoming communication tape (and vice versa for messages from F to P_i). S can read the destination fields of these messages but *cannot read their contents*. S can also send messages from itself to F. When S corrupts P_i, S learns P_i's entire input and output history and writes a message to its output tape notifying Z of the corruption. P_i can no longer be activated. When S halts or enters its waiting state, if S has delivered a message to $Q \in \{P_1, ..., P_n, F\}$ in this activation then S activates Q. Otherwise S activates Z.

2.2.4 Definition of UC Security

Universally composable security means that no environment can "tell the difference" between a real model protocol execution and an ideal model protocol execution. Because the environment only interacts with a protocol through its inputs and outputs, the main way the environment can distinguish between the two models is through communication with the adversary. We can prevent this by guaranteeing that every real world adversary can be emulated by some ideal world adversary. More specifically, protocol π *UC realizes* ideal functionality F if for any real adversary A, there exists an ideal adversary S such that no environment Z, with any input, can distinguish between A and π in the real world and S and F in the ideal world. If this holds, no advantage can be gained by using the real protocol over the (trivially secure) ideal protocol. We formally state this as follows:

Definition 2.2.1 *(UC Security):*

Let $REAL_{\pi,A,Z}(m, z)$ be the probability distribution of environment Z's output in the real model with protocol π, adversary A, security parameter m, initial input z, and uniformly generated random tape input for all parties. Let

$REAL_{\pi,A,Z}$ be the ensemble $\{REAL_{\pi,A,Z}(m,z)\}_{m\in\mathbb{N},z\in\{0,1\}^*}$.

Let $IDEAL_{F,S,Z}(m,z)$ be the probability distribution of environment Z's output in the ideal model with ideal functionality F, ideal adversary S, security parameter m, initial input z, and uniformly generated random tape input for all parties. Let $IDEAL_{F,S,Z}$ be the ensemble $\{IDEAL_{F,S,Z}(m,z)\}_{m\in\mathbb{N},z\in\{0,1\}^*}$.

*Protocol π **UC realizes** ideal functionality F if for any real adversary A, there exists an ideal adversary S such that for all environments Z, $REAL_{\pi,A,Z} \approx IDEAL_{F,S,Z}$.*

Generally speaking, protocols are proven UC secure by demonstrating that no real adversary A can learn anything that some ideal adversary S can't also learn. This is not trivial because the ideal model and real model produce very different protocol messages. For example, consider the environment that asks the adversary to report every single protocol message that it sees. While A sees the real protocol's full (and possibly complicated) message transcript, S only sees messages being forwarded to and from the ideal functionality. If both S and A report their views honestly, the environment can easily distinguish between them. Therefore, S has to "fill in" the missing protocol messages by simulating them internally. The challenge of producing a valid security proof is showing an S that can do this in spite of its limited power for interference in the ideal setting. Furthermore, S cannot use rewinding in its simulation (this is a considerable restriction). If, in spite of all these restraints, S can still match any message that A can produce, then clearly A has very limited adversarial power with respect to the real protocol.

2.2.5 The Hybrid Computation Model

Before we can state the UC composition theorem, we must define a third model of computation: the *hybrid model*. The hybrid model is a straightforward mix of the real and ideal models. It is exactly the same as the real model except that all parties also have access to an ideal functionality F. That is, the only difference between the hybrid and real models is that hybrid parties $P_1, ..., P_n$ can communicate with F as well as with each other. Likewise, the hybrid adversary can read the contents of "real" messages between two parties P_i and P_j but cannot read "ideal" messages between P_i and F.

Hybrid parties may access an unlimited number of F instances. In other words, a hybrid computation includes not just one trusted party but an arbitrary number of trusted parties (each modeled as a distinct ITM with its own state and tapes). Each instance of F has its own *session id*, a unique identity that distinguishes it from all other instances. All messages to and from an F instance generally include that instance's session id.

For any F, the hybrid model with ideal access to F is known as the *F-hybrid model*. A hybrid model may have multiple functionalities. For example, the hybrid model with ideal functionalities F, G, and H is known as the *(F,G,H)-hybrid model*

We can convert the F-hybrid model into the real model by replacing all references to F with references to an equivalent real protocol π. This works as follows: when a hybrid party would send a message to an F instance, we have it supply the message as input to a corresponding invocation of π. When a hybrid party would receive a message from an F instance, we have it read the output from the corresponding invocation of π. See [7] for a more detailed description.

2.2.6 The UC Composition Theorem

The UC composition theorem guarantees that secure protocols remain secure under general concurrent composition. When protocol π composes, we are saying that it runs within some larger system that uses π multiple times in arbitrary ways. We can view the larger system as another protocol (call it Ω).[9] If π UC realizes ideal functionality F, secure composition means that there is no difference between Ω using F and Ω using π. In other words, we want to show equivalence between the F-hybrid model and the real model that replaces F with π. This is exactly what the composition theorem does. We state it as follows:

Theorem 2.2.1 *(The UC composition theorem):*

Let F be an ideal functionality and let π be a protocol that UC realizes F. Let Ω be some protocol that operates in the F-hybrid model. Let Ω^π be the same protocol in the real model, where F is replaced with π. Let $HYBRID^F_{\Omega,H,Z}(m, z)$ be the probability distribution of environment Z's output in the F-hybrid model with protocol Ω, hybrid adversary H, security parameter m, initial input z, and uniformly generated random tape input for all parties. Let $HYBRID^F_{\Omega,H,Z}$ be the ensemble $\{HYBRID^F_{\Omega,H,Z}(m, z)\}_{m\in\mathbb{N},z\in\{0,1\}^*}$.

For any real adversary A, there exists a hybrid adversary H such that for all environments Z, $REAL_{\Omega^\pi,A,Z} \approx HYBRID^F_{\Omega,H,Z}$.

Proof: see [7] for a detailed proof.

An important corollary of the theorem allows us to design secure protocols in the F-hybrid model, then replace F with any protocol that UC realizes F

[9]Even if the system consists of many unrelated protocols, they can be thought of as components of a broader umbrella protocol.

without losing security. This allows for simple and straightforward protocol design. The corollary states:

Corollary 2.2.1 *For ideal functionalities F and G, let α be a protocol that UC realizes G in the F-hybrid model. Let π be a protocol that UC realizes F in the real model. Let α^π be the real version of α where F is replaced with π. Then α^π UC realizes G in the real model.*

Proof ([7]): The general theorem guarantees that there is some adversary H in the F-hybrid model such that $REAL_{\alpha^\pi,A,Z} \approx HYBRID^F_{\alpha,H,Z}$ for any environment Z and any real adversary A. The security of α in the F-hybrid model guarantees that there is some ideal adversary S in the ideal model such that $HYBRID^F_{\alpha,H,Z} \approx IDEAL_{G,S,Z}$ for any environment Z and any hybrid adversary H. We thus have that $REAL_{\alpha^\pi,A,Z} \approx HYBRID^F_{\alpha,H,Z} \approx IDEAL_{G,S,Z}$, i.e. $REAL_{\alpha^\pi,A,Z} \approx IDEAL_{G,S,Z}$. By the definition of UC security, this means that α^π UC realizes G in the real model. $\square$

2.3 Some Ideal Functionalities

In this section we define a simple example ideal functionality that illustrates the UC framework. We then define standard ideal functionalities for bit commitment and oblivious transfer. These functionalities are vital to our results.

2.3.1 An Example: Ideal Salary Comparison

Consider the example from the introduction. Two parties P_1 and P_2 want to compare their salaries without revealing them. Figure 2.1 defines an ideal functionality that solves this problem. We call this functionality $F_{COMPARE}$.

Functionality $F_{COMPARE}$

Parties: P_1 *with salary* x_1*,* P_2 *with salary* x_2*, ideal adversary* S

On receiving message $(\mathbf{salary}, \mathbf{sid}, \mathbf{x_1})$ from P_1 and message $(\mathbf{salary}, \mathbf{sid}, \mathbf{x_2})$ from P_2, set a according to

$$a = 1 \text{ if } x_1 > x_2$$

$$a = 2 \text{ if } x_2 > x_1$$

$$a = 0 \text{ if } x_1 = x_2$$

and write the message $(\mathbf{comparison}, \mathbf{sid}, \mathbf{a})$ to both P_1 and P_2. After this, halt (ignore all future messages).

Figure 2.1: An ideal functionality for two-party salary comparison

An ideal computation with P_1 and P_2, ideal functionality $F_{COMPARE}$, ideal adversary S, and environment Z proceeds as follows:

1. Z sets $x_1 = P_1$*'s salary* and $x_2 = P_2$*'s salary.*
2. Z chooses a session id sid and writes the message $(\mathbf{salary}, \mathbf{sid}, \mathbf{x_1})$ to P_1, who forwards this message to $F_{COMPARE}$.
3. Z writes the message $(\mathbf{salary}, \mathbf{sid}, \mathbf{x_2})$ to P_2, who forwards this message to $F_{COMPARE}$.
4. $F_{COMPARE}$ selects an answer a as described in figure 2.1 and sends the message $(\mathbf{comparison}, \mathbf{sid}, \mathbf{a})$ to P_1 and P_2.
5. P_1 and P_2 forward this response to their respective outputs, which Z reads.

We note some consequences of this definition. First, $F_{COMPARE}$ is a single-use functionality, as it permanently halts after performing a single comparison. If we wish to make n comparisons, we must do so in a hybrid model by using n $F_{COMPARE}$ instances. Second, $F_{COMPARE}$ does not reveal its output to the ideal adversary S. This maintains output privacy, but output privacy isn't very important here because all parties receive the same output. We may wish to emphasize this fact by having $F_{COMPARE}$ also send its response to S. Many practical functionalities do this. Third, an ideal computation produces no output if a malicious P_i withholds its $(\mathbf{salary}, \mathbf{sid}, \mathbf{x_i})$ message from $F_{COMPARE}$, thus keeping the ideal functionality in a perpetual waiting state. We can avoid this by having $F_{COMPARE}$ send an empty $(\mathbf{receipt}, \mathbf{sid})$ message to all parties after receiving any input. This would guarantee output as long as at least one party is honest. Other modifications are possible depending on our requirements.

Obviously our choice of definition critically affects the security of our protocols. When we call a protocol universally composable, all we are saying is that it is as secure as some ideal functionality. This may not be a valuable claim if our ideal functionality is poorly defined. For example, we can easily modify $F_{COMPARE}$ to write $(\mathbf{comparison}, \mathbf{sid}, \mathbf{a}, \mathbf{x_1}, \mathbf{x_2})$ as its response to P_1 and P_2. But while this trivially remains secure by definition, it hardly remains secure in any intuitive sense. So we must take special care that our ideal functionalities capture our natural notions of security.

2.3.2 Ideal Bit Commitment

Bit commitment is a two-stage interaction between a *committer* P_i and a *receiver* P_j. P_i *commits* to a bit b by running some algorithm C with P_j and

Functionality F_{COM}

Parties: committer P_i with input bit b, receiver P_j, ideal adversary S

Commitment: On receiving message $(\mathbf{commit}, \mathbf{sid}, \mathbf{b})$ from P_i, write message $(\mathbf{receipt}, \mathbf{sid})$ to both P_j and S. Ignore any subsequent commit messages.

Decommitment: On receiving message $(\mathbf{decommit}, \mathbf{sid})$ from P_i, if a commitment has already been made to some bit b then write message $(\mathbf{open}, \mathbf{sid}, \mathbf{b})$ to both P_j and S and halt. Otherwise, halt.

Figure 2.2: An ideal functionality for bit commitment

decommits to b (*opens* b) by running some algorithm D that reveals b. A valid commitment process must be *binding* and *hiding*. Binding means that after committing to b, P_i cannot decommit to $\overline{b}$. Hiding means that before decommitment, P_j cannot guess the value of b with greater probability than was possible before the commitment began.

The ideal functionality for commitment is known as F_{COM} and has been defined in numerous papers (i.e. [10], [7]). All of our protocols rely on access to ideal bit commitment. We reproduce F_{COM} in figure 2.2.

Note that this definition provides the ideal adversary S with any message that P_j receives. Also, this is a single-use functionality (each F_{COM} instance

Functionality F_{OT}

Parties: sender P_i *with input bits* b_0 *and* b_1*, receiver* P_j *with input bit* v*, ideal adversary* S

Transfer: On receiving message $(\mathbf{input}, \mathbf{sid}, \mathbf{b_0}, \mathbf{b_1})$ from P_i and message $(\mathbf{choice}, \mathbf{sid}, \mathbf{v})$ from P_j, write message $(\mathbf{received}, \mathbf{sid}, \mathbf{b_v})$ to P_j and message $(\mathbf{receipt}, \mathbf{sid})$ to P_i and S. Ignore all future inputs.

Figure 2.3: an ideal functionality for oblivious transfer

handles only a single commitment and decommitment).

2.3.3 Ideal Oblivious Transfer

1-out-of-2 oblivious transfer (OT) is an interaction between a *sender* P_i and a *receiver* P_j. Given inputs bits b_0, b_1 for P_i and input bit v for P_j, an OT reveals b_v to P_j while revealing nothing about $b_{\overline{v}}$. Furthermore, P_i learns nothing about v. *1-out-of-4 oblivious transfer* is the same protocol, but with P_i providing four inputs for P_j to select from instead of two.

The ideal functionality for oblivious transfer is known as F_{OT} and has been defined in numerous papers (i.e. [7], [30]). Our COT protocol relies on access to ideal oblivious transfer. We reproduce F_{OT} in figure 2.3.

Note that this definition reveals b_v to P_j but not to the ideal adversary S. This models the intuition that P_j's choice remains secret even after the protocol completes. Like F_{COM}, this is a single-use functionality.

Chapter 3

Committed Oblivious Transfer

This chapter reviews the abstract committed oblivious transfer (COT) primitive and the COT implementation from [18]. Our main results in the following chapters consist of translating this protocol into the UC framework.

3.1 Informal Definition

Committed oblivious transfer (COT) is a two-party interaction between a *sender* T and a *receiver* R. In short, COT is a straightforward combination of bit commitment and oblivious transfer. T starts with commitments to bits b_0 and b_1. R starts with a commitment to bit v. After T and R run the protocol, R is committed to b_v while knowing nothing about $b_{\overline{v}}$. Furthermore, T knows nothing about v. COT was first defined by Crépeau, van de Graaf, and Tapp in [18] (earlier defined as "verifiable oblivious transfer" in [17]).

The advantage of committed oblivious transfer over traditional oblivious transfer is that an OT protocol provides no guarantee that its input bits are what they are claimed to be. For example, imagine that b_0 and b_1 are the results of some earlier computation that both parties were involved in. When

R receives b_v, R expects to receive one of the results from that computation. But nothing prevents T from replacing b_0 and b_1 with arbitrary bits. R must verify T's inputs through some external means. In contrast, COT guarantees that neither party can change its inputs after commitments have been made. If T and R end the earlier computation by committing to its results, they cannot change these values when doing subsequent transfers.

The above COT description is also known as *1-out-of-2 COT*. *1-out-of-4* COT is defined analogously to 1-out-of-4 OT.

3.2 The CGT Implementation

[18] describes a protocol for implementing COT using underlying bit commitment and oblivious transfer primitives (we call this the *CGT protocol*). We review relevant fundamentals and reproduce the protocol description in the following subsections:

3.2.1 Codes

The CGT protocol uses codes. We briefly review codes as they apply to our work.

For a finite set S, a *block code of length n* (also known simply as a *code*) is a non-empty subset of S^n. Say that $S = \{a, b, c, d, e, f\}$. An example of a block code is the repetition code of length 3, represented by the set $\{aaa, bbb, ccc, ddd, eee, fff\}$. We *encode* the value $w \in S$ by mapping it into some $c \in C$ and *decode* $c \in C$ by reverse-mapping it back to w. w is known as an *information word* and c is known as a *codeword*. For example, we can use the repetition code of length 3 to encode the information word "e" into the codeword "*eee*".

A code from $S = \{0, 1\}$ is known as a *binary code*. The binary repetition code of length 3 is the set $\{000, 111\}$.

A code of length n produces codewords with n symbols. The *distance*[1] between two codewords is the number of spots in which their symbols differ. The *minimum distance* for a code C is the smallest distance between any two codewords in C. An *error correcting* code is a code that has some capacity to fix improperly formed codewords. Consider the code $C = \{00000, 11100, 01111\}$. Codewords 00000 and 01111 have distance 4. The minimum distance for C is 3. Using the "nearest codeword" correction strategy, we can correct the value 00001 to produce 00000.

Let F_2 be the field with elements 0 and 1, as standardly defined. A binary *linear code of length* n is a subspace of F_2^n. An $[n, k]$ *linear code* is a linear code of length n with dimension k. We can encode for a linear code through use of a *generator matrix*. For an $[n, k]$ linear code C, a $k \times n$ matrix G is a generator matrix for C if G's rows are linearly independent and the row space of G is C. We encode a k-bit information word w by representing it as a $1 \times k$ vector and multiplying it by a generator matrix.

An $[n, k, d]$ linear code is an $[n, k]$ linear code with minimum distance d.

We can choose a random linear code by selecting a random generator matrix. That is, a random $k \times n$ matrix with linearly independent rows generates some $[n, k]$ linear code.

See [49] for a more detailed overview on coding theory.

[1] also known as *Hamming distance*.

3.2.2 Privacy Amplification

The CGT protocol also uses *privacy amplification* ([6]). Roughly speaking, a privacy amplification function $f : \{0,1\}^n \rightarrow \{0,1\}^r$ has the feature that for any $x \in \{0,1\}^n$, a party that only knows some of x has no knowledge of $f(x)$. Such functions are useful for converting partially secret data into fully secret data.

Privacy amplification can be achieved through the use of *universal hash functions* ([13]). A universal hash function is a randomly chosen function from a *universal class*, defined as follows:

Definition 3.2.1 *A class C of functions $\{0,1\}^n \rightarrow \{0,1\}^r$ is universal if for f chosen randomly from C and for any distinct $x_1, x_2 \in \{0,1\}^n$, the probability that $f(x_1) = f(x_2)$ is at most $\frac{1}{2^r}$.*

An example of a universal class is the class of linear functions from $\{0,1\}^n$ to $\{0,1\}^r$ ([13]).

The CGT protocol uses privacy amplification functions where $r = 1$, i.e. functions that map an n-bit string into a single bit. This means a random linear function is the cumulative XOR of a random subset of an input string's bits. For example, assume that $n = 5$ and party V knows some (but not all) of $x = x_1x_2x_3x_4x_5$. We choose a random linear function f by selecting a random value $s \in \{0,1\}^5$. Say that $s =$ "01010". This means $f(x) = x_2 \oplus x_4$. Say that $s =$ "11001". This means $f(x) = x_1 \oplus x_2 \oplus x_5$. As long as s is chosen randomly, V knows nothing about $f(x)$ except with negligible probability.

3.2.3 Bit Commitment with XOR

The CGT protocol requires the ability to prove arbitrary XOR relationships between committed bits without revealing their contents. That is, given commitments to bits $b_1, ...b_n$, we must be able to prove $b_1 \oplus ... \oplus b_n = x$ (for some publicly known x) without revealing any of $b_1, ..., b_n$.[2]

We do this by using a *BCX* (described by Kilian in [41], Rudich and Bennett as mentioned in [18]). A BCX is a special type of commitment that supports XOR proofs. It can be constructed from any regular commitment scheme C as follows: to commit to bit b with security parameter m, choose m bits $b_{1L}, ..., b_{mL}$ randomly. Then, for $i : 1 \le i \le m$, set $b_{iR} = b_{iL} \oplus b$. Finally, use C to commit to each of the $2m$ bits $b_{1L}, b_{1R}, ..., b_{mL}, b_{mR}$ (we identify a commitment to b' with the label $\boxed{b'}$). To decommit, simply decommit to all $2m$ bits. The verifier accepts the decommitment if for all $i : 1 \le i \le m$, $b_{iL} \oplus b_{iR} = b$ for the same value b.

Say that T has made BCX commitments to bits b and d (as shown in figure 3.1).

$$\begin{array}{llll} b = & \boxed{b_{1L}} \oplus \boxed{b_{1R}}, & d = & \boxed{d_{1L}} \oplus \boxed{d_{1R}}, \\ & \boxed{b_{2L}} \oplus \boxed{b_{2R}}, & & \boxed{d_{2L}} \oplus \boxed{d_{2R}}, \\ & \quad \cdots & & \quad \cdots \\ & \boxed{b_{mL}} \oplus \boxed{b_{mR}} & & \boxed{d_{mL}} \oplus \boxed{d_{mR}} \end{array}$$

such that for $i : 1 \le i \le m$, $b_{iL} \oplus b_{iR} = b$ and $d_{iL} \oplus d_{iR} = d$.

Figure 3.1: Two BCX commitments to bits b and d

T can prove to V that $b \oplus d = x$ through the following interaction:

1. **V**: choose random permutations π_b, π_d that shuffle the rows of b and d respectively and send $(\pi_\mathbf{b}, \pi_\mathbf{d})$ to T.

[2] Note that proving the XOR of two bits shows whether or not the bits are equal.

2. **T**: For (shuffled) rows $i : 1 \leq i \leq m$, calculate $L_i = b_{iL} \oplus d_{iL}$ and $R_i = b_{iR} \oplus d_{iR}$. Send $(\mathbf{L_1}, \mathbf{R_1}, \mathbf{L_2}, \mathbf{R_2}, ..., \mathbf{L_m}, \mathbf{R_m})$ to V.

3. **V**: For (shuffled) rows $i : 1 \leq i \leq m$, randomly choose $choice_i \in_r \{L, R\}$. Send $(\mathbf{choice_1}, \mathbf{choice_2}, ..., \mathbf{choice_m})$ to T.

4. **T**: For $i : 1 \leq i \leq m$, if $choice_i = L$, then open $\boxed{b_{iL}}$ and $\boxed{d_{iL}}$. Else, open $\boxed{b_{iR}}$ and $\boxed{d_{iR}}$.

5. **V**: For $i : 1 \leq i \leq m$, if $choice_i = L$, check that $b_{iL} \oplus d_{iL} = L_i$ (act similarly if $choice_i = R$). Accept the proof if these equations always check out.

For any i, $b \oplus d = (b_{iL} \oplus b_{iR}) \oplus (d_{iL} \oplus d_{iR}) = (b_{iL} \oplus d_{iL}) \oplus (b_{iR} \oplus d_{iR}) = L_i \oplus R_i$. Therefore, $b \oplus d = x$ if and only if $L_i \oplus R_i = x$. So V only needs to verify that L_i and R_i were chosen correctly to find the proof convincing. V accomplishes this by having T decommit to either b_{iL} and d_{iL} or b_{iR} and d_{iR} for each BCX row, thus verifying one of L_i or R_i. While the other value remains unverified, T can only cheat undetected with probability $1/2$, since T doesn't know which choice V will make. Over all m rows, T can successfully cheat with probability $1/2^m$.

When T wants to prove $x = b^1 \oplus ... \oplus b^n$ for n bits $b^1, ..., b^n$, the above technique works with the change that $L_i = b^1_{iL} \oplus ... \oplus b^m_{iL}$ and $R_i = b^1_{iR} \oplus ... \oplus b^m_{iR}$.

A BCX for bit b can be safely used with at most one proof. This is because after a proof finishes, V knows either b_{iL} or b_{iR} for each row i. In any subsequent proof, V can learn the other value and thus learn b. So if we want to use a BCX for multiple proofs, we must "copy" it first. This is done by having T generate $6m$ new b_{iL}, b_{iR} values for b (in $3m$ rows). V then sends T a permutation that shuffles these rows. After shuffling, the first m rows are labeled b_p, the second m rows are labeled b', and the third m rows are labeled b''. Note that each of b_p, b', and b'' is a valid BCX. T proves equality between b

and b_p with an XOR proof. This makes b and b_p ineligible for future proofs, but b' and b'' remain untouched. And because V had T shuffle the rows randomly, the probability that $b' \neq b$ or $b'' \neq b$ is negligible in m.

3.2.4 Informal Protocol Description

Say that T has BCX commitments to bits b_0 and b_1 and R has a BCX commitment to bit v. A first attempt at performing a COT may have T run a plain OT with R and then have R commit to b_v. However, this approach is problematic. As mentioned earlier, it is impossible to verify that T and R don't replace their input bits with arbitrary bits that have nothing to do with b_0, b_1, and v.

We can resolve this problem by having T and R transfer m-bit strings (for security parameter m) instead of single bits. That is, T commits to random strings $s_0 = s_0^1 s_0^2 ... s_0^m$ and $s_1 = s_1^1 s_1^2 ... s_1^m$, then for $i : 1 \leq i \leq m$ runs an OT with R with bits s_0^i and s_1^i. T then maps these strings to the original bits b_0 and b_1 with a privacy amplification function. This means R can only learn b_v by learning all of the bits of s_v, except with negligible probability. This approach offers effective techniques (described below) for verifying that both parties act honestly.

However, we still face a subtle security problem. R can only verify the bits it receives from the OT. The unreceived bits can't be verified because R never sees them in the first place. T can exploit this and learn v as follows: T correctly feeds the bits of s_0 into each OT while replacing the bits of s_1 with arbitrary values. T then announces the privacy function correctly. If R chooses to learn b_0, verification checks succeed and the protocol runs correctly. If R chooses to learn b_1, verification checks fail and R aborts the protocol. T

learns v simply by seeing whether or not R aborts.

We can resolve this through the use of error-correcting codes. For a code with minimum distance d, T transfers random codewords c_0 and c_1 to R instead of arbitrary strings. R chooses to learn all the bits of c_v and a few of the bits of $c_{\overline{v}}$. This allows R to verify both of T's inputs. If d is sufficiently large, T cannot transfer either codeword incorrectly without being detected by R while R will not know enough bits of $c_{\overline{v}}$ to learn anything about $b_{\overline{v}}$.

This is precisely how the CGT protocol works. In this protocol, T commits to two random codewords, uses oblivious transfer to transmit one of these codewords to R, and announces a privacy amplification function that maps the codewords to b_0 and b_1. R then commits to b_v. Both parties engage in a number of proofs along the way to verify their actions. Specifically, T and R interact as follows (note that we omit some important details to keep this description informal):

1. R picks and announces a random error-correcting $[m,k,d]$ linear code where $k > (1/2 + 2\sigma)$ and $d > \epsilon m$ for positive constants σ and ϵ.

2. T picks two random codewords $c_0, c_1 \in C$ and commits to each through BCX commitments on their bits. That is, for $c_0 = c_0^1 c_0^2 ... c_0^m$, T commits to c_0^i for $i : 1 \leq i \leq m$ (and does the same for c_1).

3. R randomly selects disjoint sets $I_{\overline{v}}, I_v \subset \{1, .., m\}$ where $|I_{\overline{v}}| = |I_v| = \sigma m$. For $i : 1 \leq i \leq m$, T and R run an oblivious transfer with c_0^i and c_1^i. R chooses to receive $c_{\overline{v}}^i$ for $i \in I_{\overline{v}}$ and c_v for all other indices. This means that R learns most of the bits of c_v and a few of the bits of $c_{\overline{v}}$.

4. R announces $I = I_v \bigcup I_{\overline{v}}$. For $i \in I$, T decommits to both c_0^i and c_1^i. This lets R verify that at least $2\sigma m$ of T's OT inputs were chosen honestly according to protocol. This also lets R know all m bits of c_v. Furthermore, because d is linear in m, R can use error correction to correctly learn c_v even if T feeds some of the unverified bits

(i.e. the bits with indices $i \notin I$) into the OT incorrectly.[3]

5. T randomly picks and announces a privacy amplification function h and announces bits x_0, x_1 such that $h(c_0^1...c_0^m) \oplus x_0 = b_0$ and $h(c_1^1...c_1^m) \oplus x_1 = b_1$. Since R fully knows c_v, R learns b_v. Since R only partially knows $c_{\overline{v}}$, R knows nothing about $b_{\overline{v}}$.

6. R makes a BCX commitment to b_v.

A few concerns remain unresolved. T must prove to R that c_0 and c_1 are in fact codewords (and not just arbitrary strings), since R relies on the fact that they are codewords to learn b_v. T must also prove that h, x_0, and x_1 have the required relationships. R must prove to T that it has learned b_v and not $b_{\overline{v}}$. R must also prove that it finally commits to b_v and not $\overline{b_v}$.

All of these proofs can be accomplished through XOR proofs on appropriate BCX commitments. We leave a more formal description of the protocol, including details of the XOR proofs, to chapter 5.

[3]More precisely, because d is linear in m, T would have to feed a linear number of bits into the OT incorrectly for R to receive the wrong codeword. However, T doesn't yet know I when it performs the transfer. So at least one of these bits will likely be chosen for decommitment when R announces I, at which point R can detect any inconsistencies. The probability that this does not happen is negligible in m.

Chapter 4

Universally Composable Bit Commitment with XOR

In this chapter we define F_{BCX}, the ideal functionality for bit commitment with XOR proofs. We then formally define the hybrid BCX protocol as described in chapter 3 and prove that it UC realizes F_{BCX} (assuming hybrid access to F_{COM}). In the next chapter we define a UC COT protocol that runs in the F_{BCX}-hybrid model. We only consider the two-party setting, as extending to the multi-party setting is simple and straightforward (see chapter 6 for further discussion).

4.1 Ideal BCX Functionality

The ideal two-party functionality for bit commitment with XOR is known as F_{BCX}. It supports an unlimited number of commitments and decommitments between a designated committer and receiver. It also supports an unlimited number of XOR proofs on committed bits, so long as these bits have not been decommitted. Each commitment has a unique identifier known as its *cid*. Each F_{BCX} instance has a unique identifier known as its *sid*. F_{BCX} is defined in figure 4.1.

4.2 Hybrid BCX Protocol

The hybrid protocol for BCX follows the description from chapter 3 and runs in the F_{COM}-hybrid model. Given committer P, receiver V, hybrid adversary A, environment Z, and security parameter m, the protocol runs as follows:

Commitment:

1. Z provides P with input $(\mathbf{commit}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$.
2. P checks that cid is a new value (has not been used for any previous commitment). If so, P generates m random bits $b_{1L}, ..., b_{mL}$ and computes $b_{iR} = b \oplus b_{iL}$ for $1 \leq i \leq m$. Otherwise, P aborts the operation.
3. P invokes $2m$ new F_{COM} instances (with sids $cid_{1L}, cid_{1R}, ..., cid_{mL}, cid_{mR}$). P gives instance cid_{iL} the input message $(\mathbf{commit}, \mathbf{cid_{iL}}, \mathbf{b_{iL}})$ and instance cid_{iR} the input message $(\mathbf{commit}, \mathbf{cid_{iR}}, \mathbf{b_{iR}})$.
4. Each F_{COM} instance responds with a receipt. For $\alpha \in \{L, R\}$, instance $cid_{i\alpha}$ writes the message $(\mathbf{receipt}, \mathbf{cid_{i\alpha}})$ to V and A.
5. V, upon receiving all $2m$ receipts, outputs $(\mathbf{receipt}, \mathbf{sid}, \mathbf{cid})$.

Decommitment:

1. Z provides P with input $(\mathbf{decommit}, \mathbf{sid}, \mathbf{cid})$.
2. P checks that cid refers to a commitment to some bit b that has never been decommitted. If so, P decommits to all $2m$ underlying bits of b by giving F_{COM} instance $cid_{i\alpha}$ the input $(\mathbf{decommit}, \mathbf{cid_{i\alpha}})$ for $\alpha \in \{L, R\}$ and $1 \leq i \leq m$. Otherwise, P aborts the operation.

Functionality F_{BCX}

Parties: (dummy) committer and prover P_I, (dummy) receiver and verifier V_I, ideal adversary S

Commitment: On receiving message $(\mathbf{commit}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$ from P_I, check that cid is a new value (i.e. has not been used for any previous commitment). If so, store b and write message $(\mathbf{receipt}, \mathbf{sid}, \mathbf{cid})$ to both V_I and S. Otherwise, do nothing.

Decommitment: On receiving message $(\mathbf{decommit}, \mathbf{sid}, \mathbf{cid})$ from P_I, check that cid refers to an existing commitment to some bit b that has never been decommitted. If so, write message $(\mathbf{open}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$ to both V_I and S. Otherwise, do nothing.

XOR Proof: On receiving message $(\mathbf{prove}, \mathbf{sid}, \mathbf{x}, \mathbf{cid^1}, ..., \mathbf{cid^n})$ from P_I, check that each cid^i refers to a commitment to some bit b_i that has never been decommitted. If this holds and $b_1 \oplus ... \oplus b_n = x$, write message $(\mathbf{proof}, \mathbf{sid}, \mathbf{x}, \mathbf{cid^1}, ..., \mathbf{cid^n})$ to V_I and S. Otherwise, do nothing.

Figure 4.1: An ideal functionality for bit commitment with XOR

3. Each F_{COM} instance $cid_{i\alpha}$ responds with the message $(\mathbf{open}, \mathbf{cid_{i\alpha}}, \mathbf{b_{i\alpha}})$ to V and A.

4. V verifies that for each i, $b_{iL} \oplus b_{iR} = b$ (for some b), and if so outputs $(\mathbf{open}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$. Otherwise, V outputs nothing.

XOR Proof:

1. Z provides P with input $(\mathbf{prove}, \mathbf{sid}, \mathbf{x}, \mathbf{cid^1}, ..., \mathbf{cid^n})$.

2. P checks that each cid^i refers to a commitment to some bit b_i where none of its $2m$ underlying bits have been decommitted and that $b_1 \oplus ... \oplus b_m = x$. If any of these conditions fails, P aborts the operation.

Copy Phase

3. P chooses, for $1 \leq i \leq 3m$, random bit $b_{1_{iL}}$ and computes $b_{1_{iR}} = b_1 \oplus b_{1_{iL}}$.

4. P invokes $6m$ new F_{COM} instances (supporting $3m$ bit-pairs, with base sid pid) and gives instance $pid^1_{i\alpha}$ the input $(\mathbf{commit}, \mathbf{pid^1_{i\alpha}}, \mathbf{b_{1_{i\alpha}}})$.

5. Each F_{COM} instance $pid^1_{i\alpha}$ responds with the message $(\mathbf{receipt}, \mathbf{pid^1_{i\alpha}})$ for V and A.

6. V produces a random permutation π of these $3m$ rows and sends message $(\mathbf{permute}, \mathbf{sid}, \mathbf{pid}, \mathbf{cid^1}, \pi)$ to P.

7. P and V label the first m permuted pairs as b'_1, the second m pairs as b''_1, and the third m pairs as b'''_1. For $1 \leq i \leq m$ and $\alpha \in \{L, R\}$, they assign $cid^{1'}_{i\alpha} = pid^1_{\pi^{-1}(i)\alpha}$, $cid^{1''}_{i\alpha} = pid^1_{\pi^{-1}(i+m)\alpha}$, and $cid^{1'''}_{i\alpha} = pid^1_{\pi^{-1}(i+2m)\alpha}$ (where $\pi^{-1}(i)$ is the inverse of the permutation of i).[1]

8. Steps 3 - 7 are repeated for the other committed bits $b_2, ..., b_n$.

Prove Phase ($\boxed{b_j} = \boxed{b'_j}$ *for* $1 \leq j \leq n$*)*

9. P calculates $L_i = b_{1_{iL}} \oplus b'_{1_{iL}}$ and $R_i = b_{1_{iR}} \oplus b'_{1_{iR}}$ for $1 \leq i \leq m$ and sends message $(\mathbf{announce}, \mathbf{sid}, \mathbf{pid}, \mathbf{cid^1}, \mathbf{L_1}, \mathbf{R_1}, ..., \mathbf{L_m}, \mathbf{R_m})$ to V.

[1] This is simply a cid labelling scheme that lets us treat b'_1, b''_1, and b'''_1 as standard BCX commitments, where $cid^{1'}_{i\alpha}$ refers to bit $b'_{1_{i\alpha}}$, etc.

10. V randomly selects $choice_i \in_r \{L, R\}$ for $1 \leq i \leq m$ and sends message $(\mathbf{choices}, \mathbf{sid}, \mathbf{pid}, \mathbf{cid^1}, \mathbf{choice_1}, ..., \mathbf{choice_m})$ to P.

11. For $1 \leq i \leq m$ and $choice_i = \alpha$, P opens $b_{1_{i\alpha}}$ and $b'_{1_{i\alpha}}$ by sending inputs $(\mathbf{decommit}, \mathbf{cid^1_{i\alpha}})$ and $(\mathbf{decommit}, \mathbf{cid^{1'}_{i\alpha}})$ to the appropriate F_{COM} instances.

12. The appropriate F_{COM} instances write messages $(\mathbf{open}, \mathbf{cid^1_{i\alpha}}, \mathbf{b_{1_{i\alpha}}})$ and $(\mathbf{open}, \mathbf{cid^{1'}_{i\alpha}}, \mathbf{b'_{1_{i\alpha}}})$ to V and A.

13. V checks for each $choice_i = \alpha$ and $\Theta_i = (L_i \text{ if } \alpha_i = L \text{ or } R_i \text{ if } \alpha_i = R)$ that $\Theta_i = b_{1_{i\alpha}} \oplus b'_{1_{i\alpha}}$. If this fails for any i, V aborts the protocol.

14. Steps 9 - 13 are repeated for the other committed bits $b_2, ..., b_n$.

Reassignment Phase

At this point, the original commitments to $b_1, ..., b_n$ have been "used up" because some of their underlying bits have been decommitted. This makes them unsuitable for future proofs.[2] P and V get around this by "reassigning" the underlying bits. For $1 \leq i \leq m$, $1 \leq j \leq n$, and $\alpha \in \{L, R\}$, whenever the protocol instructs P to decommit to $b_{j_{i\alpha}}$, P sends the message $(\mathbf{decommit}, \mathbf{cid^{j'''}_{i\alpha}})$ to F_{COM} instance $cid^{j'''}_{i\alpha}$ (and never sends any more messages to F_{COM} instance $cid^j_{i\alpha}$). Likewise, V expects to receive decommitments from instance $cid^{j'''}_{i\alpha}$ instead of instance $cid^j_{i\alpha}$. This essentially replaces b_j with b'''_j. The bits for b'''_j are "fresh" and after the above prove phase b'''_j is guaranteed to equal b_j except with negligible probability ([18]).

Prove Phase ($\boxed{b''_1} \oplus ... \oplus \boxed{b''_n} = x$)

Note that after the first prove phase, $b''_j = b_j$ for $1 \leq j \leq n$ except with negligible probability ([18]). So showing $b''_1 \oplus ... \oplus b''_n = x$ is equivalent to showing $b_1 \oplus ... \oplus b_n = x$.

15. V chooses random permutations $\pi''_1, ..., \pi''_n$ that shuffle the rows of $b''_1, ..., b''_n$ and sends message $(\mathbf{permute3}, \mathbf{sid}, \mathbf{pid}, \pi''_\mathbf{1}, ..., \pi''_\mathbf{n})$ to P.

[2] Say that for some i, V has decided to learn $b_{1_{iL}}$ in the current proof. In any future proof, V can choose to learn $b_{1_{iR}}$, which reveals b_1.

16. P calculates $L_i = b''_{1_{iL}} \oplus ... \oplus b''_{n_{iL}}$ and $R_i = b''_{1_{iR}} \oplus ... \oplus b''_{n_{iR}}$ for $1 \leq i \leq m$ and sends message $(\mathbf{announce2}, \mathbf{sid}, \mathbf{pid}, \mathbf{L_1}, \mathbf{R_1}, ..., \mathbf{L_m}, \mathbf{R_m})$ to V.

17. V randomly selects $choice_i \in_r \{L, R\}$ for $1 \leq i \leq m$ and sends message $(\mathbf{choices2}, \mathbf{sid}, \mathbf{pid}, \mathbf{choice_1}, ..., \mathbf{choice_m})$ to P.

18. For $1 \leq i \leq m$, if $choice_i = L$, P opens $b''_{1_{iL}}, ..., b''_{n_{iL}}$ by sending inputs $(\mathbf{decommit}, \mathbf{cid^{1''}_{iL}})$, ..., $(\mathbf{decommit}, \mathbf{cid^{n''}_{iL}})$ to the appropriate F_{COM} instances. If $choice_i = R$, P instead opens $b''_{1_{iR}}, ..., b''_{n_{iR}}$ the same way.

19. The appropriate F_{COM} instances write messages $(\mathbf{open}, \mathbf{cid^{1''}_{i\alpha}}, \mathbf{b''_{1_{i\alpha}}})$, ..., $(\mathbf{open}, \mathbf{cid^{n''}_{i\alpha}}, \mathbf{b''_{n_{i\alpha}}})$ to V and A.

20. V checks for each $choice_i$ that if $choice_i = L$, then $L_i = b''_{1_{iL}} \oplus ... \oplus b''_{n_{iL}}$ (and similarly if $choice_i = R$). V also checks that for $1 \leq i \leq m$, $L_i \oplus R_i = x$. If all conditions hold, V outputs $(\mathbf{proof}, \mathbf{sid}, \mathbf{x}, \mathbf{cid^1}, ..., \mathbf{cid^n})$. Otherwise, V outputs nothing.

Note that after an XOR proof completes, no b'_j or b''_j value may be used in any future decommitment or proof. These should be thought of as "temporary values" that have no scope beyond the current proof.

4.3 UC Security of the Hybrid Protocol

We now state the following theorem:

Theorem 4.3.1 *The above protocol UC realizes F_{BCX} in the F_{COM}-hybrid model.*

Proof: In order to prove this theorem, we must show that for any adversary A interacting with the hybrid protocol in the F_{COM}-hybrid model, there is an adversary S interacting with F_{BCX} in the ideal model such that no environment Z can distinguish between the two models under any input. We construct such an S as follows:

In general, S runs a simulated copy of A within its code (call this A_{SIM}). All inputs from Z are forwarded to A_{SIM}'s virtual input. All virtual outputs of A_{SIM} are forwarded to S's actual output.

When A_{SIM}, having virtually corrupted a party, wants to use its ideal functionality F_{COM}, S plays the role of that ideal functionality. This gives S the considerable power of receiving simulated commitments, learning their contents, and opening them however it chooses.

We will first show security for a single commitment/decommitment process, then for a single XOR proof, and finally when the protocol is used for multiple operations.

4.3.1 Security of a Single Commit/Decommit Operation

Say that some environment Z runs the protocol to perform a single commitment and decommitment with bit b. As mentioned above, S internally runs a copy of A_{SIM}, forwarding inputs from Z to A_{SIM}'s virtual input and forwarding virtual outputs from A_{SIM} to its own output. The remaining details of the simulation depend on who is corrupted and who is honest. We consider each corruption scenario in turn:

Corrupted P, Uncorrupted V

The Simulation:

S plays the role of F_{COM} for A_{SIM}. For commitment, an honest hybrid V would only output a receipt after receiving $2m$ F_{COM} receipts. So S follows A_{SIM}'s code and keeps track of all bits b_{iL}, b_{iR} that A_{SIM} virtually sends to F_{COM} in the name of T. After $2m$ such commitments have been made, S

determines b by calculating $b = b_{iL} \oplus b_{iR}$ (for any i) and has P_I send the message $(\mathbf{commit}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$ to F_{BCX}.

For decommitment, an honest hybrid V would only output its result after all $2m$ committed bits have been correctly decommitted. S follows A_{SIM}'s code and waits until A_{SIM} has sent virtual decommit messages to all $2m$ F_{COM} copies. S then verifies that for all i, $b_{iL} \oplus b_{iR} = b$. If so, S has P_I send the message $(\mathbf{decommit}, \mathbf{sid}, \mathbf{cid})$ to F_{BCX}. Otherwise, S does nothing.

Proof of Security:

S can play the role of F_{COM} perfectly, and since the protocol only involves messages from P and F_{COM}, S does not have to simulate any messages from an honest V.

If A is semi-honest, the ideal simulation is clearly identical to an actual hybrid interaction. In both computations, V will output $(\mathbf{receipt}, \mathbf{sid}, \mathbf{cid})$ after commitment and $(\mathbf{open}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$ after decommitment. And since both A and A_{SIM} are given the same input bit b (provided by Z), we know that all b_{iL}'s and b_{iR}'s will be computed the same way.

If A is malicious, it may compute the b_{iL}'s and b_{iR}'s incorrectly. First, it may not compute each b_{iL} randomly as required. But since A_{SIM} follows the same code as A, any non-randomness will be perfectly emulated in the ideal simulation. Second, A may choose its bits inconsistently, such that there exist two i and j where $b_{iL} \oplus b_{iR} \neq b_{jL} \oplus b_{jR}$. In the hybrid interaction, this would cause commitment to complete successfully but decommitment to fail (V fails to produce output when it recognizes the inconsistency). In the ideal simulation, S doesn't check for consistency before committing but does check before decommitting. Any inconsistency causes S to withhold its decommit message to F_{BCX}, which means V_I produces no output.

We thus have a perfect simulation for this scenario.

Uncorrupted P, Corrupted V

The Simulation:

S plays the role of both F_{COM} and an uncorrupted hybrid P for A_{SIM}. For commitment, P_I makes a commitment using F_{BCX} in the ideal world. S is notified that this commitment occurred, but does not know the committed bit b. S then simulates $2m$ F_{COM} receipts for A_{SIM} (note that F_{COM} commitment receipts don't include any information on the committed bits, so S can produce valid receipts even without knowing b). Once A_{SIM} instructs V to output its receipt, S has V_I output its receipt.

For decommitment, P_I sends a decommit message to F_{BCX}, which then opens the commitment and reveals b to S. S then simulates $2m$ decommit messages from F_{COM} as follows: for F_{COM} instance iL, S randomly chooses b_{iL} and simulates a decommitment to this value. For F_{COM} instance iR, S calculates $b_{iR} = b \oplus b_{iL}$ and simulates a decommitment to b_{iR}. When A_{SIM} instructs V to output its receipt, S has V_I output the same value.

Proof of Security:

The critical difference between the ideal and hybrid interactions is that a hybrid P commits to bits b_{iL} and b_{iR} such that $b_{iL} \oplus b_{iR} = b$, whereas the simulated P cannot do this because S doesn't know b. However, this has no effect on the adversary's view of the commitment messages. A cannot see the contents of any message sent to F_{COM} and F_{COM} receipts (which A can see) contain no information about b. So S can simulate the commitment messages perfectly.

Upon decommitment, S learns the value of b before having to simulate any decommit messages. Thus, S can choose b_{iL} and b_{iR} according to an honest P's program and decommit to these values. The fact that simulated commitments have already been made is not a problem. Since S plays the role of F_{COM} for A_{SIM}, S can decommit to any value it chooses without producing any message inconsistencies.

If A is malicious, it may have V produce incorrect output. S easily emulates this by producing V_I's output according to A_{SIM}'s code.

We thus have a perfect simulation for this scenario.

Both Parties Corrupted

The entire protocol is a deterministic functionality of P's and V's inputs and random tapes. When both parties are corrupted, S has full access to all of this information and can thus emulate the hybrid interaction exactly.

Neither Party Corrupted

When neither party is corrupted, A passively sees a complete message transcript between P and V. S must simulate this transcript for A_{SIM}. This is easy to do by following the simulation from the *Uncorrupted P, Corrupted V* scenario. Security follows from the security of that scenario. A malicious A may also interfere with the protocol by interrupting message delivery, possibly preventing V from producing output. S can simulate this perfectly by not delivering messages from F_{BCX} to V_I when A_{SIM} does something that would block V's output.

Dynamic Corruption

We've now demonstrated the (perfect) UC security of the hybrid protocol for commitment and decommitment in the presence of static, malicious adversaries. However, we must also demonstrate security against adaptive adversaries that may corrupt parties at any point in the protocol. The added challenge is that a newly corrupted party's input must be consistent with its previous protocol messages. For example, when P is uncorrupted, S simulates commitment messages from P to b without actually knowing b. As long as P remains uncorrupted and the simulated messages are indistinguishable from real messages, security is maintained. But if P is later corrupted, the adversary now learns b. It does not in general hold that the earlier messages remain indistinguishable given this new knowledge. We must explicitly guarantee this and do so as follows:

Corrupting P:

Say that A corrupts P at some point in a hybrid interaction. If this corruption occurs during a commitment, then P has sent somewhere between 1 and $2m$ commitment messages of the forms $(\mathbf{commit}, \mathbf{cid_{iL}}, \mathbf{b_{iL}})$ and $(\mathbf{commit}, \mathbf{cid_{iR}}, \mathbf{b_{iR}})$ to F_{COM}. S has simulated these messages without knowing b. However, since messages to F_{COM} are private and secure, A has not seen the b_{iR} or b_{iL} values in these messages. The remaining message data is independent of b. If the corruption occurs during a decommitment, P's decommit messages are entirely independent of P's private inputs (note that when the simulation for decommitment begins, S learns the correct value of b in the ideal interaction, so b is no longer private). In either case, revealing P's input produces no inconsistencies with the existing message transcript. So S can corrupt P_I and make P_I's actual input available to A_{SIM} in the simulation.

Corrupting V:

V receives no private inputs in this protocol; its output fully depends on the messages received from F_{COM}. So V can trivially be corrupted at any point without adverse security consequences.

This completes the security proof for a single commitment/decommitment process.

4.3.2 Security of a Single XOR Proof

Now say that Z runs the protocol to perform a single XOR proof on committed bits $b_1, ..., b_n$ and result x. S internally runs a copy of A_{SIM}, forwarding inputs from Z to A_{SIM}'s virtual input and forwarding virtual outputs from A_{SIM} to its own output. The rest of the simulation runs as follows:

Corrupted P, Uncorrupted V

The Simulation:

S plays the role of both F_{COM} and an uncorrupted hybrid V for A_{SIM}. For the copy phase, S follows A_{SIM}'s code to generate the appropriate commitments for b_1. After A_{SIM} has virtually produced $6m$ such commitments, S chooses a random permutation π and simulates a **permute** message from V to P. This process is repeated for b_2 through b_n.

For the prove phase between b_j and b'_j, S simulates messages from V by following an honest V's program. Note that V has no private inputs, so S does not need additional information to run V's code correctly. S simulates messages from P by following A_{SIM}'s code. At the end of this phase, S follows V's code to determine if the proof was conducted properly. If not, S withholds all messages from P_I to F_{BCX} in the ideal interaction.

The rest of the protocol involves a reassignment phase and another prove phase. The reassignment phase produces no messages, so can be trivially simulated perfectly. S simulates the prove phase in exactly the same way that it simulates the first prove phase. If the entire simulation completes without complaint by simulated V, S has P_I send the message $(\mathbf{prove}, \mathbf{sid}, \mathbf{x}, \mathbf{cid^1}, ..., \mathbf{cid^n})$ to F_{BCX} to complete the ideal proof and have V_I generate output.

Proof of Security:

The copy phase is essentially the same as a commitment process. There are only two differences: the copy phase uses $6m$ bits instead of $2m$ bits and the copy phase involves the additional steps of V choosing a random permutation and P rearranging its bit-rows accordingly. The first difference has no security consequences. The second difference creates no problems because S generates a random permutation the same way an actual hybrid V would. Thus, security of the copy phase follows from the security of commitment, which has already been shown.

In the prove phases, V has no private inputs and produces all of its messages exclusively by tossing random coins. So S can easily simulate V's messages perfectly. S's only other responsibility is to guarantee that V_I produces output only when V would produce output (i.e. only when the hybrid proof would succeed).

For the prove phase that shows $b''_1 \oplus ... \oplus b''_n = x$, assume that the relation is correct. If A_{SIM} is semi-honest, P follows the protocol correctly, so the hybrid protocol is guaranteed to succeed. S completes the proof in the ideal interaction accordingly. If A_{SIM} is malicious, P may attempt to cheat. If S finds that V would detect this, S withholds proof completion in the ideal interaction. Note that S judges success based on what V would see, not on what S sees itself. This is because there is a negligible probability that P can cheat without V noticing (see [18] for details). S would notice such cheating, since S knows simulated P's private inputs, but it must still complete the ideal proof consistent with how the hybrid proof would complete.

Now assume that the relation is false (i.e. $b''_1 \oplus ... \oplus b''_n \neq x$). A semi-honest A_{SIM} has P generate each L_i and R_i such that $L_i \neq R_i$. Both S and V can detect this and produce protocol failure. If A_{SIM} is malicious, there is a negligible probability that P can still provide a convincing proof for V.[3] In such a case, the hybrid protocol would incorrectly report success. S cannot emulate this through F_{BCX}, which refuses to complete an equality proof for incorrect relations.

We use the same analysis for the prove phase between b_j and b'_j. Thus, S simulates the hybrid interaction perfectly except with a negligible probability of failure, i.e. S produces an indistinguishable simulation for this scenario.

Uncorrupted P, Corrupted V

The Simulation:

S plays the role of both F_{COM} and an uncorrupted hybrid P for A_{SIM}. For the copy phase, S does not know the values of $b_1, ..., b_n$, so it simulates fake commitment receipts with no underlying values. S then follows A_{SIM}'s code

[3]For any i, P can announce L_i correctly and R_i incorrectly. If V chooses to open the left bits, V will not detect the cheating. This happens with probability $\frac{1}{2}$. Over all i, this happens with probability $\frac{1}{2^m}$.

to generate V's **permute** messages.

For the prove phase between b_j and b'_j, S follows A_{SIM}'s code to generate V's simulated messages. But since S doesn't know the value of b_j and b'_j, S cannot follow an honest P's code to simulate P's messages. Rather, S simulates the $(\mathbf{announce}, \mathbf{sid}, \mathbf{pid}, \mathbf{cid^j}, \mathbf{L_1}, \mathbf{R_1}, ..., \mathbf{L_m}, \mathbf{R_m})$ message by choosing each L_i uniformly at random and setting $R_i = L_i$. After A_{SIM} responds with the message $(\mathbf{choices}, \mathbf{sid}, \mathbf{pid}, \mathbf{cid^j}, \mathbf{choice_1}, ..., \mathbf{choice_m})$, S simulates P's selective decommitments as follows: Say that $choice_i = L$. S chooses bit p uniformly at random. If L_i was announced as 0, S sets bit $q = p$. If L_i was announced as 1, S sets $q = \overline{p}$. S then simulates the F_{COM} decommitment of $b_{j_{iL}}$ to p and $b'_{j_{iL}}$ to q. If $choice_i = R$, S acts analogously.

S simulates the final prove phase in the same way. Finally, S waits for A_{SIM} to instruct V to produce its output, at which point S delivers the ideal proof message from P_I to F_{BCX} to complete the ideal proof.

Proof of Security:

As in the previous scenario, the copy phase is essentially the same as commitment, so security follows accordingly.

For the prove phase between b_j and b'_j, S does not generate the simulated L_i and R_i values as a hybrid P would. S generates each L_i completely at random and sets $R_i = L_i$. In contrast, a hybrid P generates L_i according to the equation $L_i = b_{j_{iL}} \oplus b'_{j_{iL}}$ and R_i according to the equation $R_i = b_{j_{iR}} \oplus b'_{j_{iR}}$. But since $b_{j_{iL}}$ and $b'_{j_{iL}}$ are both completely random, L_i itself is completely random. And since P is honest, R_i is guaranteed to equal L_i if b_j and b'_j are equal. So both the ideal and hybrid models produce the same exact distribution.

After A_{SIM} announces its $choice_i$ values, S must simulate decommitments to either $b_{j_{iL}}$ and $b'_{j_{iL}}$ or $b_{j_{iR}}$ and $b'_{j_{iR}}$ (for each i). If $choice_i = L$, then S simulates decommitments to two uniformly random bits p and q such that $p = q$ (if $L_i = 0$) or $p \neq q$ (if $L_i = 1$). The fact that S didn't commit to these bits in the first place poses no problems (as argued earlier). In the hybrid interaction, an honest P decommits to the actual bits $b_{j_{iL}}$ and $b'_{j_{iL}}$ that it originally committed to. Since both bits were originally chosen uniformly at random, the decommitted bits are still uniformly random subject to the restriction that they are equal (if $L_i = 0$) or different (if $L_i = 1$).

If $choice_i = R$, the ideal simulation runs in exactly the same way. The hybrid interaction, however, is not the same, since $b_{j_{iR}}$ and $b'_{j_{iR}}$ are not chosen randomly. But in this case A does not know $b_{j_{iL}}$ and $b'_{j_{iL}}$ (these values are not decommitted). Since $b_{j_{iR}} = b_j \oplus b_{j_{iL}}$ and $b'_{j_{iR}} = b'_j \oplus b'_{j_{iL}}$, the decommitted bits still appear uniformly random from A's perspective, so the probability distribution remains the same.

We apply the same analysis to the final prove phase.

A malicious A may have V produce incorrect output. S can easily emulate this by following A_{SIM}'s code and producing incorrect V_I output accordingly. A may also have V generate its protocol messages non-randomly (V's messages should be the exclusive result of random coin tosses). Again, S can emulate this by following A_{SIM}'s code. A may try to choose messages in some special way that causes P to reveal more information than S knows in the ideal interaction, but the hybrid protocol guarantees that this is impossible. This is because the probability distribution of P's messages is fully independent of V's messages. The only part of P's message transcript that depends on V is in step 12, when P decommits to either the left bits or the right bits of its commitments according to V's $choice_i$ values. But, as already argued, the decommitted values are always uniformly random from A's perspective. So A cannot extract off-limits information no matter what messages it chooses for V.

Finally, an honest P may be asked to prove that $b_1 \oplus ... \oplus b_n = x$ when in fact $b_1 \oplus ... \oplus b_n = \overline{x}$. In the ideal world, it is impossible for such a proof to succeed, since F_{BCX} ignores the request and never sends a receipt to V_I. Likewise, in the hybrid model an honest P halts the protocol without sending any messages to V.

We thus have a perfect simulation for this scenario.

Both Parties Corrupted

The entire protocol is a deterministic functionality of P's and V's inputs and random tapes. When both parties are corrupted, S has full access to all of this information and can thus emulate the hybrid interaction exactly.

Neither Party Corrupted

When neither party is corrupted, A passively sees a complete message transcript between P and V. S must simulate this transcript for A_{SIM}. We do this through a straightforward combination of the *Uncorrupted P, Corrupted V* and *Corrupted P, Uncorrupted V* simulations. If Z initiates a correct XOR proof (where the bits do in fact have the specified XOR relationship), S receives a valid proof receipt from F_{BCX}, conducts the simulation, and delivers the receipt to V_I to output. If Z initiates an incorrect XOR proof (where the bits do not have the specified XOR relationship), F_{BCX} never produces a receipt and S is never notified that a proof was even attempted. So S can't simulate any hybrid messages. However, in a hybrid interaction an honest T would catch the inconsistency and immediately abort the proof. So there would be no hybrid messages to simulate anyway.

Dynamic Corruption

Corrupting P:

Say that A corrupts P at some point in a hybrid interaction. If corruption occurs during the copy phase, security follows from the security of commitment.

If corruption occurs in the first prove phase after S has simulated an $(\mathbf{announce}, \mathbf{sid}, \mathbf{pid}, \mathbf{cid^j}, \mathbf{L_1}, \mathbf{R_1}, ..., \mathbf{L_m}, \mathbf{R_m})$ message, S has produced this message without knowing P's input bits b_j and b'_j. Thus, each L_i and R_i was chosen at random. We must ensure that these random choices are consistent with the newly revealed values of b_j and b'_j. S does so as follows: if L_i was announced as 0, then S randomly sets $b_{j_{iL}}$ and sets $b'_{j_{iL}} = b_{j_{iL}}$. If L_i was announced as 1, then S randomly sets $b_{j_{iL}}$ and sets $b'_{j_{iL}} = \overline{b_{j_{iL}}}$. In either case, S sets $b_{j_{iR}} = b_j \oplus b_{j_{iL}}$ and $b'_{j_{iR}} = b'_j \oplus b'_{j_{iL}}$. If $b_j = b'_j$, then the resultant R_i is the same as L_i (i.e. consistent with the announced value of R_i). If $b_j \neq b'_j$, then the protocol would have failed when P aborted earlier and we never get to this step in the first place.

If corruption occurs in the first prove phase after S has simulated P's selective decommitments, we face two scenarios. If $choice_i = L$, then S has decommitted to randomly chosen $b_{j_{iL}}$ and $b'_{j_{iL}}$ subject to $b_{j_{iL}} \oplus b'_{j_{iL}} = L_i$. S then simply sets $b_{j_{iR}} = b_j \oplus b_{j_{iL}}$ and $b'_{j_{iR}} = b'_j \oplus b'_{j_{iL}}$ to maintain consistency. If $choice_i = R$, then S has decommitted to randomly chosen $b_{j_{iR}}$ and $b'_{j_{iR}}$ such that $b_{j_{iR}} \oplus b'_{j_{iR}} = R_i$. S simply sets $b_{j_{iL}} = b_j \oplus b_{j_{iR}}$ and $b'_{j_{iL}} = b'_j \oplus b'_{j_{iR}}$ to maintain consistency.

If corruption occurs during the second prove phase, we apply the above analysis to maintain consistency.

Corrupting V:

Say that A corrupts V at some point in a hybrid interaction. If corruption occurs during the copy phase, security follows from the security of commitment. If corruption occurs during either prove phase, any message that S simulated in the name of V consists exclusively of random bits that use no private inputs. S can trivially maintain consistency by setting simulated V's random tape appropriately.

This completes the security proof for a single XOR proof.

4.3.3 Security under Multiple Operations

We've shown security with respect to a single operation (i.e. a single commitment, decommitment, or XOR proof), but the BCX protocol supports an unlimited number of operations within a single instance. So we need to show that the protocol retains security when used multiple times with an arbitrary scheduling scheme. For example, imagine the hybrid XOR proof without the copying and reassignment phases. This proof can securely show that two committed bits b_1 and b_2 are equal. But we lose security if we repeat the proof a second time. This is because for each i, V can assign its $choice_i$ value to the opposite of what it chose the first time, thus revealing both $b_{1_{iL}}$ and $b_{1_{iR}}$, thus revealing b_1 and b_2.

Let a single operation of the protocol consist of input β, output γ, and public message transcript δ (where "public" means viewable to the adversary). We claim that security holds under multiple operations if (β, γ, δ) provides no information to the adversary on any other operation beyond that provided by (β, γ). In other words, an operation's message transcript leaks no information that isn't also leaked from its inputs and outputs. This claim holds because our single-usage security proofs tell us that S can simulate a message transcript that is indistinguishable from an actual hybrid transcript for a single operation. So the only way a hybrid adversary can gain advantage over S is to use information from one operation's transcript to learn something about another operation. By showing that no information leaks between operations, we preclude this possibility.[4]

For commitment and decommitment, this is straightforward. The public message transcript of a commitment reveals no information whatsoever, as the only sensitive information is the $2m$ bits being committed to and these values do not appear in the transcript. Decommitment reveals the values of all $2m$ bits, but these bits have no value once they are decommitted (in particular, they are not used in other commitments and cannot be used in future XOR proofs).

For an XOR proof, the only part of P's message transcript that could reveal information is when P announces L_i and R_i values and decommits to selected bits based on V's $choice_i$ responses (the rest of P's messages are nothing more than ideal commitments through F_{COM}.). Say that for row i, P generates the messages L_i (where $L_i = b_{1_{iL}} \oplus b_{2_{iL}}$) and R_i (where $R_i = b_{1_{iR}} \oplus b_{2_{iR}}$) and decommits to $b_{1_{iR}}$ and $b_{2_{iR}}$ after receiving $choice_i = R$ from V. As already argued, L_i, R_i, $b_{1_{iR}}$, and $b_{2_{iR}}$ always appear uniformly random to any party without information on $b_{1_{iL}}$ or $b_{2_{iL}}$. Thus, knowledge about $b_{1_{iL}}$ and $b_{2_{iL}}$ is

[4] This doesn't apply to inputs and outputs because these values can leak information but do so equally in both the ideal and hybrid models. For example, if P proves that $b_1 = b_2$ and then decommits to b_1, this also reveals the value of b_2. This is true regardless of which model we're in.

required to extract information from P's messages. But due to the protocol's copy phase, each instance of an XOR proof uses unique values for $b_{1_{iL}}$ and $b_{2_{iL}}$ that are randomly generated and completely independent of all values in other instances. So it is impossible to gain this knowledge. This remains true even when P and V conduct multiple simultaneous proofs on the same bits, because P generates a new set of "fresh" bits when a proof begins (in steps 3-5). If the protocol is modified such that the reassignment phase occurs immediately after step 6 (instead of after step 14), then P and V perform this phase in the same order in which each proof completes step 6. In other words, say that proof instance ρ is the next proof to complete step 6. P and V each perform the reassignment mapping of $cid_{i\alpha}^{j}$ to ρ's version of $cid_{i\alpha}^{j'''}$ (recall that $cid_{i\alpha}^{j'''}$ is unique for each instance). This reassignment is both global and atomic: all other proof instances from this point forward now use ρ's version of $cid_{i\alpha}^{j'''}$ instead of $cid_{i\alpha}^{j}$ and no other proof instance may perform a reassignment while ρ's reassignment is ongoing. Note that the other proof instances cannot be considered fully complete until ρ finishes step 14 (because ρ's reassigned value is only guaranteed to equal the original value after step 14 completes).

V also produces messages, but they are random and rely on no private inputs, so they trivially provide no information on other operations.

This completes the proof of theorem 4.3.1. $\square$

Chapter 5

Universally Composable Committed Oblivious Transfer

In this chapter we define F_{COT}, the ideal functionality for committed oblivious transfer. We then formally define an extended version of the COT protocol described in chapter 3 and prove that it UC realizes F_{COT} (assuming hybrid access to F_{BCX} and F_{OT}). We only consider the two-party setting, as extending to the multi-party setting is simple and straightforward (see chapter 6).

5.1 Ideal Committed Oblivious Transfer

The ideal two-party functionality for COT is known as F_{COT}. It supports an unlimited number of commitments, decommitments, and transfers (in both directions). It also supports XOR proofs between three committed bits as well as AND proofs. That is, given three committed bits b_1, b_2, b_3, we can show that $b_1 \oplus b_2 = b_3$ or that $b_1 \wedge b_2 = b_3$. We add support for AND proofs because this makes F_{COT} more useful as a primitive for multi-party computation. A party conducts a proof using a binary relation Y (in the form of a truth table), where Y must be an XOR relation or an AND relation. Each commitment has a unique identifier known as its *cid*. Each F_{COT} instance has a unique identifier known as its *sid*. F_{COT} is defined in figures 5.1 and 5.2.

5.2 Hybrid COT Protocol

The hybrid protocol for COT follows the description from chapter 3 and runs in the (F_{BCX},F_{OT})-hybrid model. It supports AND proofs using a simple trick described in [18]. Say that party T is committed to bits b_1, v, and d and wants to prove to party R that $b_1 \wedge v = d$. T does this by committing to $b_0 = 0$ and performing a COT *with itself* using transfer bits b_0 and b_1 and choice bit v. After the transfer finishes, T is committed to b_v. If $v = 0$, this means that $b_v = 0 = b_1 \wedge 0$. If $v = 1$, this means that $b_v = b_1 = b_1 \wedge 1$. The proofs required for the transfer convince R that everything proceeds correctly. T then performs a conventional XOR proof to show that $b_v = d$.

Given parties T and R, hybrid adversary A, environment Z, security parameter m, and positive constants σ and ϵ (used for selecting a code in the transfer phase), the protocol runs as follows:

Commitment:

> T and R maintain two running instances of F_{BCX}: $F_{BCX}^{(T\to R)}$ and $F_{BCX}^{(R\to T)}$ (with respective sids sid' and sid''). All commitments from T to R are forwarded to $F_{BCX}^{(T\to R)}$ and all commitments from R to T are forwarded to $F_{BCX}^{(R\to T)}$.

Decommitment:

> All decommitments from T to R are forwarded to $F_{BCX}^{(T\to R)}$ and all decommitments from R to T are forwarded to $F_{BCX}^{(R\to T)}$.

Functionality F_{COT}

Parties: (dummy) participant T_I, (dummy) participant R_I, ideal adversary S. We also define the variables P_i and P_j as generic labels that can refer to either party.

Commitment: On receiving message $(\mathbf{commit}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$ from P_i, check that *cid* is a new value (i.e. has not been used for any previous commitment). If so, store b and write message $(\mathbf{receipt}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid})$ to both P_j and S. Otherwise, do nothing.

Decommitment: On receiving message $(\mathbf{decommit}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid})$ from P_i, check that *cid* refers to an existing commitment to some bit b that has never been decommitted. If so, write message $(\mathbf{open}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid}, \mathbf{b})$ to both P_j and S. Otherwise, do nothing.

Figure 5.1: An ideal functionality for committed oblivious transfer: commitment and decommitment phases

Functionality F_{COT} (continued)

Transfer:	On receiving message $(\mathbf{transfer}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid})$ from P_i, check that cid_0 and cid_1 refer to unopened commitments by P_i to respective bits b_0 and b_1, vid refers to an unopened commitment by P_j to bit v, and cid_n does not refer to an existing commitment by either party. If any of these conditions fails, do nothing. Otherwise, record a new commitment to b_v with cid cid_n, write message $(\mathbf{treceived}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid}, \mathbf{b_v})$ to P_j (but not S) and write message $(\mathbf{treceipt}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid})$ to P_i and S.
AND/XOR Proof:	On receiving message $(\mathbf{prove}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid_1}, \mathbf{cid_2}, \mathbf{cid_3}, \mathbf{Y})$, check that cid_1, cid_2 and cid_3 refer to unopened commitments by P_i to bits b_1, b_2, and b_3, respectively. Also check that Y is an XOR or AND relationship and $Y(b_1, b_2) = b_3$ holds. If all conditions hold, write message $(\mathbf{proof}, \mathbf{P_i}, \mathbf{P_j}, \mathbf{sid}, \mathbf{cid_1}, \mathbf{cid_2}, \mathbf{cid_3}, \mathbf{Y})$ to P_j and S. Otherwise, do nothing.

Figure 5.2: An ideal functionality for committed oblivious transfer: transfer and proof phases

Transfer (T to R):

1. Z provides T with input $(\mathbf{transfer}, \mathbf{T}, \mathbf{R}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid})$ (where cid_0 and cid_1 are T's respective commitments to bits b_0 and b_1, and vid is R's commitment to bit v).

Validated Codeword Generation

2. T sends message $(\mathbf{sendcode}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid})$ to R.

3. R chooses a decodable $[m, k, d]$ linear code C with $k > (1/2 + 2\sigma)$m and $d > \epsilon m$ and sends message $(\mathbf{code}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{C})$ to T.

4. T picks random codewords $c_0, c_1 \in C$ ($|c_0| = |c_1| = m$) and uses $F_{BCX}^{T \to R}$ to commit to their component bits $c_0^1, ..., c_0^m$ and $c_1^1, ..., c_1^m$. (*see section 5.2.1*)

5. T uses $F_{BCX}^{T \to R}$ to prove that c_0 and c_1 are codewords. (*see section 5.2.1*)

Validated Codeword Transfer

6. R randomly picks disjoint subsets $I_{\overline{v}}, I_v \subset \{1, .., m\}$ where $|I_{\overline{v}}| = |I_v| = \sigma m$. For $1 \leq i \leq m$, R sets $choice_i = \overline{v}$ if $i \in I_{\overline{v}}$ and $choice_i = v$ otherwise (if $i \in I_v$ or if i is outside both sets).

7. T and R invoke $F_{OT}(c_0^i, c_1^i)(choice_i)$ (m times), giving R $w^1, ..., w^m$. *(see section 5.2.1)*

8. R sets $I = I_{\overline{v}} \bigcup I_v$ and writes the message $(\mathbf{subset}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{I})$ to T.

9. For each $i \in I$, T uses $F_{BCX}^{T \to R}$ to decommit to c_0^i and c_1^i. *(see section 5.2.1)*

10. R checks that for $i \in I_{\overline{v}}$, $w^i = c_{\overline{v}}^i$ and that for $i \in I_v$, $w^i = c_v^i$. R then sets $w^i = c_v^i$ for $i \in I_{\overline{v}}$ and corrects w using the decoding algorithm (if this fails, R aborts the protocol).

11. R invokes $F_{BCX}^{R \to T}$ m times to commit to w and uses $F_{BCX}^{R \to T}$ to prove that this is a codeword. *(see section 5.2.1)*

12. T randomly picks a subset $I_T \subset \{1, ..., m\}$ of size σm such that $I_T \cap I = \emptyset$. For $i \in I_T$, T uses $F_{BCX}^{T \to R}$ to decommit to c_0^i and c_1^i. *(see section 5.2.1)*

13. T sends message $(\mathbf{subset2}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{I_T})$ to R.

14. R proves that $w^i = c_v^i$ for $i \in I_T$ using $F_{BCX}^{R \to T}$. *(see section 5.2.1)*

Committed Bit Transfer

15. T picks a random privacy amplification $h : \{0,1\}^m \to \{0,1\}$, sets $p_0 = h(c_0) \oplus b_0$ and $p_1 = h(c_1) \oplus b_1$, and sends message $(\mathbf{amplify}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{h}, \mathbf{p_0}, \mathbf{p_1})$ to R.

16. T uses $F_{BCX}^{T \to R}$ to prove $b_0 = h(c_0) \oplus p_0$ and $b_1 = h(c_1) \oplus p_1$. *(see section 5.2.1)*

17. R sets $b_v = h(w) \oplus p_v$ and uses $F_{BCX}^{R \to T}$ to commit to b_v. *(see section 5.2.1)*

18. R uses $F_{BCX}^{R \to T}$ to prove $b_v = h(w) \oplus p_v$. *(see section 5.2.1)*

19. R outputs $(\mathbf{treceived}, \mathbf{T}, \mathbf{R}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid}, \mathbf{b_v})$.

20. T outputs $(\mathbf{treceipt}, \mathbf{T}, \mathbf{R}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_0}, \mathbf{cid_1}, \mathbf{vid})$.

Transfer (R to T):

Transfers from R to T operate the same way as transfers from T to R. For simplicity's sake, the rest of this chapter assumes transfers from T to R.

AND/XOR Proof:

All XOR proofs from T to R are forwarded to $F_{BCX}^{(T \to R)}$ and all XOR proofs from R to T are forwarded to $F_{BCX}^{(R \to T)}$.

T performs an AND proof for R by running an "internal" partial transfer and verifying the results with R. This is done as follows:

1. Z provides T with input $(\mathbf{prove}, \mathbf{T}, \mathbf{R}, \mathbf{sid}, \mathbf{cid_1}, \mathbf{cid_2}, \mathbf{cid_3}, \mathbf{Y})$ (where cid_1, cid_2, and cid_3 are T's respective commitments to bits b_1, v, and d, and Y specifies the relation $b_1 \wedge v = d$).

2. T commits to $b_0 = 0$.

3. T and R conduct steps 2-5 of the transfer phase. This convinces R that T has committed to two valid codewords.

4. T and R conduct steps 11-14 of the transfer phase *with their roles reversed*. That is, T commits to the codeword $w = c_v$ in step 11 (and proves this). R picks and announces the subset in steps 12-13. T performs the decommitments in step 12 and the proof in step 14.
5. T conducts steps 15-16 of the transfer phase with b_0 and b_1.
6. T conducts steps 17-18 of the transfer phase to commit to b_v and prove its validity. This value is equal to $b_1 \wedge v$.
7. T uses $F_{BCX}^{T \to R}$ to prove that $d = b_v$.
8. T decommits to $b_0 = 0$.
9. R outputs $(\mathbf{proof}, \mathbf{T}, \mathbf{R}, \mathbf{sid}, \mathbf{cid_1}, \mathbf{cid_2}, \mathbf{cid_3}, \mathbf{Y})$.

R performs an AND proof for T analogously.

5.2.1 Transfer Phase Details

This section explains in detail how T and R conduct the proofs performed in the transfer phase.

Step 4: T commits to codewords c_0, c_1:

T commits to each codeword bit by bit (each codeword being m bits long). That is, T selects a unique base cid tid and sends $2m$ messages $(\mathbf{commit}, \mathbf{sid'}, \mathbf{tid_0^1}, \mathbf{c_0^1})$, ..., $(\mathbf{commit}, \mathbf{sid'}, \mathbf{tid_0^m}, \mathbf{c_0^m})$, $(\mathbf{commit}, \mathbf{sid'}, \mathbf{tid_1^1}, \mathbf{c_1^1})$, ..., $(\mathbf{commit}, \mathbf{sid'}, \mathbf{tid_1^m}, \mathbf{c_1^m})$ to $F_{BCX}^{T \to R}$. $F_{BCX}^{T \to R}$ sends the receipts $(\mathbf{receipt}, \mathbf{sid'}, \mathbf{tid_0^1})$, ..., $(\mathbf{receipt}, \mathbf{sid'}, \mathbf{tid_0^m})$, $(\mathbf{receipt}, \mathbf{sid'}, \mathbf{tid_1^1})$, ..., $(\mathbf{receipt}, \mathbf{sid'}, \mathbf{tid_1^m})$ to R and A.

Step 5: T proves that c_0, c_1 are codewords:

Since C is a linear code, T shows that c_0 and c_1 are codewords by showing that the syndrome of each is zero. This requires $O(m^2)$ XOR proofs using

$F_{BCX}^{T \to R}$. See [18] for further details.

Step 7: T and R perform an OT:

For all $1 \leq i \leq m$, T and R create an F_{OT} instance with sid sid_i. T sends the message $(\mathbf{input}, \mathbf{sid_i}, \mathbf{c_0^i}, \mathbf{c_1^i})$ to F_{OT}. R sends the message $(\mathbf{choice}, \mathbf{sid_i},$ $\mathbf{choice_i})$ to F_{OT}. F_{OT} sends the message $(\mathbf{received}, \mathbf{sid_i}, \mathbf{c^i_{choice_i}})$ to R and the message $(\mathbf{receipt}, \mathbf{sid_i})$ to T and A.

Step 9: T decommits to c_0^i, c_1^i for $i \in I$:

For $i \in I$, T sends the messages $(\mathbf{decommit}, \mathbf{sid'}, \mathbf{tid_0^i})$ and $(\mathbf{decommit},$ $\mathbf{sid'}, \mathbf{tid_1^i})$ to $F_{BCX}^{T \to R}$. $F_{BCX}^{T \to R}$ sends the messages $(\mathbf{open}, \mathbf{sid'}, \mathbf{tid_0^i}, \mathbf{c_0^i})$ and $(\mathbf{open}, \mathbf{sid'}, \mathbf{tid_1^i}, \mathbf{c_1^i})$ to R and A. This process occurs $(|I| = 2\sigma m)$ times.

Step 11: R commits to w and proves $w \in C$:

R selects a unique base cid wid and sends m messages $(\mathbf{commit}, \mathbf{sid''}, \mathbf{wid^1},$ $\mathbf{w^1})$,...,$(\mathbf{commit}, \mathbf{sid''}, \mathbf{wid^m}, \mathbf{w^m})$ to $F_{BCX}^{R \to T}$. $F_{BCX}^{R \to T}$ sends m receipts $(\mathbf{receipt}, \mathbf{sid''}, \mathbf{wid^1})$,...,$(\mathbf{receipt}, \mathbf{sid''}, \mathbf{wid^m})$ to T and A. R proves that w is a codeword by showing that its syndrome is zero. This requires $O(m^2)$ XOR proofs using $F_{BCX}^{R \to T}$. See [18] for further details.

Step 12: T decommits to c_0^i, c_1^i for $i \in I_T$:

For $i \in I_T$, T sends the messages $(\mathbf{decommit}, \mathbf{sid'}, \mathbf{tid_0^i})$ and $(\mathbf{decommit},$ $\mathbf{sid'}, \mathbf{tid_1^i})$ to $F_{BCX}^{T \to R}$. $F_{BCX}^{T \to R}$ sends the messages $(\mathbf{open}, \mathbf{sid'}, \mathbf{tid_0^i}, \mathbf{c_0^i})$ and $(\mathbf{open}, \mathbf{sid'}, \mathbf{tid_1^i}, \mathbf{c_1^i})$ to R and A. This process occurs $(|I_T| = \sigma m)$ times.

Step 14: R proves $w^i = c_v^i$ for $i \in I_T$:

When $c_0^i = c_1^i$, w^i reveals no information on v, so R simply decommits: R sends the message $(\mathbf{decommit}, \mathbf{sid}'', \mathbf{wid^i})$ to $F_{BCX}^{R \to T}$ and $F_{BCX}^{R \to T}$ sends the message $(\mathbf{open}, \mathbf{sid}'', \mathbf{wid^i}, \mathbf{w^i})$ to T and A.

When $c_0^i = 0$ and $c_1^i = 1$, this means that $c_v^i = v$. So R shows that $w^i = v$ by sending the message $(\mathbf{prove}, \mathbf{sid}'', \mathbf{0}, \mathbf{wid^i}, \mathbf{vid})$ to $F_{BCX}^{R \to T}$, which responds with the message $(\mathbf{proof}, \mathbf{sid}'', \mathbf{0}, \mathbf{wid^i}, \mathbf{vid})$ for T and A.

When $c_0^i = 1$ and $c_1^i = 0$, this means that $c_v^i = \overline{v}$. So R shows that $w^i = \overline{v}$ by sending the message $(\mathbf{prove}, \mathbf{sid}'', \mathbf{1}, \mathbf{wid^i}, \mathbf{vid})$ to $F_{BCX}^{R \to T}$, which responds with the message $(\mathbf{proof}, \mathbf{sid}'', \mathbf{1}, \mathbf{wid^i}, \mathbf{vid})$ for T and A.

In total, R sends $|I_T| = \sigma m$ messages to $F_{BCX}^{R \to T}$.

Step 16: T **proves** $b_0 = h(c_0) \oplus p_0, b_1 = h(c_1) \oplus p_1$**:**

We assume h is a universal hash function defined as a random subset $\{i^1, i^2, ..., i^r\}$ (for some r) of its input bits summed together modulo 2. Therefore, $h(c_0) = c_0^{i^1} \oplus c_0^{i^2} \oplus ... \oplus c_0^{i^r}$. So T proves that $h(c_0) \oplus b_0 = p_0$ by sending the message $(\mathbf{prove}, \mathbf{sid}', \mathbf{p_0}, \mathbf{tid_0^{i^1}}, \mathbf{tid_0^{i^2}}, ..., \mathbf{tid_0^{i^r}}, \mathbf{cid_0})$ to $F_{BCX}^{T \to R}$, which responds with the message $(\mathbf{proof}, \mathbf{sid}', \mathbf{p_0}, \mathbf{tid_0^{i^1}}, \mathbf{tid_0^{i^2}}, ..., \mathbf{tid_0^{i^r}}, \mathbf{cid_0})$ for R and A. T repeats this for c_1 and b_1.

Step 17: R **commits to** $b_v = h(w) \oplus p_v$**:**

R commits to bit b_v by sending the message $(\mathbf{commit}, \mathbf{sid}'', \mathbf{cid_n}, \mathbf{b_v})$ to $F_{BCX}^{R \to T}$. $F_{BCX}^{R \to T}$ sends the message $(\mathbf{receipt}, \mathbf{sid}'', \mathbf{cid_n})$ to T and A.

Step 18: R **proves** $b_v = h(w) \oplus p_v$**:**

This follows the same logic as the proofs for steps 14 and 16. If $p_0 = p_1$, R uses the step 16 proof technique to show that $h(w) \oplus b_v = p_0$ (or equivalently

$h(w) \oplus b_v = p_1$). If $p_0 = 0$ and $p_1 = 1$, this means that $h(w) \oplus b_v \oplus v = 0$, so R proves this with the step 14 proof technique. If $p_0 = 1$ and $p_1 = 0$, this means that $h(w) \oplus b_v \oplus v = 1$, so R proves this with the step 14 proof technique.

5.3 UC Security of the Hybrid Protocol

We now state the following theorem:

Theorem 5.3.1 *The above protocol UC realizes F_{COT} in the (F_{BCX},F_{OT})-hybrid model.*

Proof: In order to prove this theorem, we must show that for any adversary A interacting with the protocol in the (F_{BCX}, F_{OT})-hybrid model, there is an adversary S interacting with F_{COT} in the ideal model such that no environment Z can distinguish between the two models under any input. We construct such an S as follows:

In general, S runs a simulated copy of A within its code (call this A_{SIM}). All inputs from Z are forwarded to A_{SIM}'s virtual input. All of A_{SIM}'s virtual outputs are forwarded to S's actual output.

When A_{SIM}, having virtually corrupted a party, wants to use its ideal functionalities F_{BCX} or F_{OT}, S plays the roles of these ideal functionalities. This gives S the power of receiving simulated commitments, learning their contents, decommitting however it chooses, faking XOR proofs, and learning the full contents of any OT transfer.

We will first briefly consider security for commitments / decommitments / XOR proofs, then show security for a single transfer, then consider security for a single AND proof, and finally show security for multiple operations.

5.3.1 Security of Commitments, Decommitments, and XOR Proofs

Since the hybrid protocol forwards commitments, decommitments, and XOR proofs directly to appropriate F_{BCX} instances, these operations are secure by definition.

5.3.2 Security of a Single Transfer

Say that Z runs the protocol to perform a single transfer from T to R, where T has committed inputs b_0 and b_1 and R has committed input v. S internally runs a copy of A_{SIM}, forwarding inputs from Z to A_{SIM}'s virtual input and forwarding virtual outputs from A_{SIM} to its own output. The rest of the simulation runs as follows:

Corrupted T, Uncorrupted R

The Simulation:

S plays the roles of F_{BCX}, F_{OT}, and an uncorrupted R for A_{SIM}. In general, S follows A_{SIM}'s code, simulating messages from R by running an honest R's program with input $v = 0$ (since S doesn't know the actual value of v). Whenever A_{SIM} conducts a proof from T, S uses R's code to check the proof's validity. If all proofs succeed (i.e. R would be convinced of the protocol's overall validity), S delivers T_I's input in the ideal world to F_{COT}, which sends an appropriate response to R_I. Otherwise, S delivers nothing to F_{COT}, thus preventing R_I from producing output. S finally has T_I output whatever A_{SIM} would have T output.

Proof of Security:

S must simulate R's messages without knowing R's private input. This is easy to do perfectly. In steps 3 and 8, R's messages consist of randomly chosen values that make no use of its private inputs. In steps 11, 14, 17, and 18, R's messages consist exclusively of commitments or XOR proofs using $F_{BCX}^{R \to T}$. While these messages involve R's private inputs, their contents are hidden (since the messages are ideal) and the headers can easily be simulated. Finally, in step 7, R provides input to F_{OT}. This is simulatable for the same reason.

S must also simulate messages from $F_{BCX}^{T \to R}$, $F_{BCX}^{R \to T}$, and F_{OT}. S simulates $F_{BCX}^{T \to R}$ messages by verifying A_{SIM}'s inputs to $F_{BCX}^{T \to R}$ and generating appropriate responses. In other words, if A_{SIM}'s inputs would be acceptable to $F_{BCX}^{T \to R}$, S simulates an appropriate response. If A_{SIM}'s inputs would not be acceptable, S withholds any response. S doesn't have the same luxury for $F_{BCX}^{R \to T}$, where R's inputs to the ideal functionality are unknown. But since R is honest, S can simply assume R's inputs are valid and generate $F_{BCX}^{R \to T}$ responses accordingly. F_{OT} simulation is a straightforward combination of the other two cases.

This results in a perfect simulation if A is semi-honest. If A is malicious, it may generate c_0, c_1 as invalid codewords, pick the subset I_T non-randomly, generate h incorrectly, or use F_{OT} with incorrect inputs. For the first and third cases, T must prove that its computations are correct using $F_{BCX}^{T \to R}$. Since $F_{BCX}^{T \to R}$ is ideal, these proofs are perfectly complete and sound, i.e. T can't cheat. So a hybrid R would always detect malicious behavior and abort the protocol, and S can likewise abort the ideal interaction. For the second case, choosing I_T non-randomly has no effect on the protocol because R's subsequent messages look the same for any I_T such that $I_T \cap I = \emptyset$.

For the fourth case, A may have T provide arbitrary inputs to each F_{OT} instead of c_0^i and c_1^i as required. As a result, R may receive an arbitrary codeword c_y that, when applied to h, causes R to commit to some b_y that has no relation to b_0 or b_1. S cannot match this in the ideal interaction, where F_{COT} guarantees that R_I receives b_0, b_1, or nothing at all. However, this can only occur if the verification in steps 8 - 10 succeeds in spite of the malicious inputs. [18] shows that this happens with negligible probability in m.

We thus have an indistinguishable simulation (a perfect simulation that fails with negligible probability) for this scenario.

Uncorrupted T, Corrupted R

The Simulation:

S plays the roles of F_{BCX}, F_{OT}, and an uncorrupted T for A_{SIM}. In the ideal interaction, F_{COT} informs R_I that bit b_v has been transferred (since S controls R_I, S also learns b_v). F_{COT} also generates a receipt for T_I, but S does not deliver this message just yet. In the simulation, S follows A_{SIM}'s code and simulates messages from T by running an honest T's program with inputs b_v (which S knows) set to the actual value of b_v and $b_{\overline{V}}$ (which S doesn't know) set to 0. Whenever A_{SIM} has R conduct a proof, S uses T's code to check the proof's validity. If all proofs succeed (i.e. T would accept the interaction), S finally delivers F_{COT}'s receipt to T_I in the ideal interaction. Otherwise, S never delivers the receipt and T_I outputs nothing. S then has R_I output whatever A_{SIM} would have R output.

Proof of Security:

S must simulate T's messages while only knowing one of T's private inputs. For the first 14 steps of the protocol, S can easily do this perfectly. This is because these messages have nothing to do with T's private inputs. Rather, they all rely on codewords that T chooses completely at random. S must also simulate messages from $F_{BCX}^{T \to R}$, $F_{BCX}^{R \to T}$, and F_{OT}, but, as already argued, this is easy to do. In step 16, S must simulate a proof from T that $b_0 = h(c_0) \oplus p_0$ and $b_1 = h(c_1) \oplus p_1$. Although this step involves T's private inputs, it consists entirely of XOR proofs using $F_{BCX}^{T \to R}$. These proofs can easily be faked because the messages involved contain no information on b_0 or b_1.

This leaves us with step 15, the only part of the protocol not covered by the above analysis. In this step, T announces a random privacy amplification function h and values p_0, p_1 such that $b_0 = h(c_0) \oplus p_0$ and $b_1 = h(c_1) \oplus p_1$. As described in the simulation, S generates these messages under the assumption that $b_{\overline{v}} = 0$. Thus, S generates $p_{\overline{v}}$ such that $p_{\overline{v}} = h(c_{\overline{v}}) \oplus 0$. At this point in the protocol, R is guaranteed not to know $c_{\overline{v}}$ except with negligible probability in m ([18]). Because h is a privacy amplification function, it follows that R has no information on $h(c_{\overline{v}})$. So the probability distribution for $p_{\overline{v}}$ is the same regardless of whether $b_{\overline{v}} = 0$ or $b_{\overline{v}} = 1$.

We thus have an indistinguishable simulation (a perfect simulation that fails with negligible probability) for this scenario.

Both Parties Corrupted

The entire protocol is a deterministic functionality of T's and R's inputs and random tapes. When both parties are corrupted, S has full access to all of this information and can thus emulate the hybrid interaction exactly.

Neither Party Corrupted

When neither party is corrupted, A passively sees a complete message transcript between T and R. S must simulate this transcript for A_{SIM}. We do this through a straightforward combination of the *Corrupted* T*, Uncorrupted* R and *Uncorrupted* T*, Corrupted* R scenarios. However, in this case S must simulate T's messages without knowing either of T's input bits (as opposed to knowing b_v but not $b_{\overline{v}}$). So S runs the simulation assuming $b_0 = 0$ and $b_1 = 0$. The analysis for the *Uncorrupted* T*, Corrupted* R scenario remains valid even with this difference. A may also try to interfere with the protocol by interrupting message delivery, possibly preventing T and R from producing output. S can easily simulate this by following A_{SIM}'s actions and only delivering ideal messages if A_{SIM} allows the simulated protocol to complete.

Dynamic Corruption

Corrupting T:

Say that A corrupts T at some point in a hybrid interaction. All of T's commitment, XOR proof, and OT messages to an ideal functionality reveal no information on T's private input, so they do not need to be justified once the input is known. If corruption occurs after steps 9 or 12, S has simulated T's decommitments to some bits of c_0 and c_1. Since c_0 and c_1 are chosen randomly, S justifies the decommitted values by setting simulated T's random tape appropriately (note that known methods exist to generate uniformly random codewords from uniformly random input bits in an easily reversible way). If corruption occurs after step 13, S has simulated T's announcement of a subset I_T. This subset is also chosen randomly and is handled the same way.

If corruption occurs after step 15, S has simulated T's announcement of a random privacy amplification function h and values p_0, p_1 such that $p_0 = h(c_0) \oplus b_0$ and $p_1 = h(c_1) \oplus b_1$. If R was corrupted at the time, then S simulated these messages with the correct value of b_v and with $b_{\overline{v}}$ assumed to be 0. If R was not corrupted, S simulated these messages by assuming that both b_0 and b_1 are 0. After corruption of T, S learns the correct values of both bits and must justify that the simulated messages are consistent with these values.

We consider the case for $b_{\overline{v}}$: If $b_{\overline{v}} = 0$, S's assumption regarding $b_{\overline{v}}$ was correct, so the simulated messages are consistent by default. If $b_{\overline{v}} = 1$, it follows that $p_{\overline{v}} \neq h(c_{\overline{v}}) \oplus b_{\overline{v}}$. This is a problem. But most of the bits of $c_{\overline{v}}$ were unknown to the adversary at the point of corruption. At most, T has revealed $\frac{m}{2} + \frac{3\sigma m}{2}$ bits from the OT in step 7, decommitments in step 9, and decommitments in step 12. This is because if any more bits had been revealed, R would not have learned enough bits of c_v to successfully complete the proof in step 14 with non-negligible probability (as shown in [18]). Therefore, at least $m - (\frac{m}{2} + \frac{3\sigma m}{2}) = \frac{m}{2} - \frac{3\sigma m}{2} \geq \frac{m}{8}$ bits of $c_{\overline{v}}$ are still unknown. If S can generate a different codeword c_S such that $h(c_S) \oplus p_{\overline{v}} = b_{\overline{v}}$ and the known bits in $c_{\overline{v}}$ are the same in c_S, this produces the required consistency between $b_{\overline{v}}$ and the simulated messages for T.

Let ℓ be the number of known bits of $c_{\overline{v}}$ and let $I_\ell \subset \{1, ..., m\}$ be the set of indices specifying the positions of these bits in the codeword (where $|I_\ell| = \ell$). Because C is a linear code, it has a generator matrix G such that $c_{\overline{v}} = wG$ for some binary k-bit information word w (where both $c_{\overline{v}}$ and w are represented as vectors). Each codeword bit $c^i_{\overline{v}}$ (for index $i \in \{1, ..., m\}$) is determined by multiplying w by the i'th column in G. Let G_i be the i'th column in G. Since

ℓ of the bits of $c_{\overline{v}}$ have been revealed, this means that the set of information words producing codewords consistent with these values is defined as all k-bit vectors w that satisfy $wG_i = c_{\overline{v}}^i$ for $i \in I_\ell$. This is simply a series of ℓ linear equations with at least $2^{n-\ell} \geq 2^{\frac{m}{8}}$ solutions (note that because $k > (\frac{1}{2}+2\sigma)m$, we know that $k > \ell$).

S proceeds as follows: S finds a random solution w_S from the above linear system and calculates $c_S = w_S G$. If $p_{\overline{v}} = h(c_S) \oplus b_{\overline{v}}$, then S sets simulated T's random tape such that c_S would be its chosen codeword for $b_{\overline{v}}$. Otherwise, S finds another solution to the linear system and tries again. The privacy amplification properties of h guarantee that for each solution w_S, $p[h(w_S G) = p_{\overline{v}} \oplus b_{\overline{v}}] = \frac{1}{2}$. So after m tries, S is guaranteed to find a solution that works except with probability $\frac{1}{2^m}$.[1]

The same analysis applies to b_v if it was unknown before corruption.

Corrupting R:

Say that A corrupts R at some point in a hybrid interaction. Aside from R's messages to an ideal functionality, R sends two messages to T. If corruption occurs after step 3, S has simulated R's announcement of a randomly chosen code C. This message is the exclusive result of random coin tosses, so S justifies it with an appropriately set random tape. If corruption occurs after step 8, S has simulated R's announcement of the subset I to T. This message is also the result of random coin tosses and is handled the same way.

[1]If we treat h as the cumulative XOR of a random subset I_h of the bits of $c_{\overline{v}}$, there is a chance that $I_h \subseteq I_\ell$. In such a case, it holds that for all valid solutions c_S, $h(c_S) = h(c_{\overline{v}})$. This would be a problem. But the probability that this happens is $\frac{1}{2^{m-\ell}} \leq \frac{1}{2^{m/8}}$ (because for each $i \notin I_\ell$, the probability that $i \in I_h$ is $\frac{1}{2}$).

5.3.3 Security of a Single AND Proof

Note that an AND proof is simply a stripped down version of a transfer. The main difference is that when T performs an AND proof, it doesn't need to execute steps 6-10 of a transfer because it already knows both codewords (although we could keep these steps in if we wanted to). Therefore, security follows from the security of a transfer operation.

5.3.4 Security under Multiple Operations

We now show security under multiple operations. Because commitment, decommitment, and XOR proofs are handled directly through F_{BCX} functionalities, we only need to consider the transfer and AND proof operations.

When T and R perform a transfer, the only point where T could possibly provide information useful to other operations is when T announces p_0 and p_1 in step 15. This is because all other messages that T produces are determined exclusively by codewords c_0 and c_1, which are randomly chosen for each transfer. So the codewords for a transfer have no meaning outside the scope of that particular transfer. In step 15, T's messages partially depend on the private values b_0 and b_1, which may be used among multiple operations. However, these messages reveal no information on b_0 and b_1 unless the adversary learns the corresponding codewords c_0 and c_1. The only way this can happen is if the adversary controls R, thus learning c_v. This reveals b_v, but this value is revealed by the protocol's output anyway.

As for R, R's messages in steps 3 and 8 are the results of completely random coin tosses and thus trivially reveal no information useful to other operations. The rest of R's messages consist of invocations of ideal functionalities where the message contents don't appear in the protocol's public message transcript.

The above analysis also works for AND proofs due to their similar structure.

This completes the proof of theorem 5.3.1. $\square$

Chapter 6

Universally Composable Multi-Party Computation

In this chapter we show how to achieve UC two-party and multi-party computation using UC COT. This essentially consists of plugging our COT protocol into the circuit evaluation structure specified in [30].

6.1 Two-Party Computation

6.1.1 1-out-of-4 COT

Two-party computation can be achieved with 1-out-of-4 COT (defined by the ideal functionality F_{4COT}). This is defined as F_{COT} with the change that sender T provides four committed input bits to a transfer and receiver R provides two committed input bits. After the transfer completes, R is committed to exactly one of T's inputs. We show how to realize F_{4COT} in the F_{COT}-hybrid model:

Commitment, Decommitment, AND/XOR Proofs:

T and R share a single F_{COT} instance and forward these tasks directly to that instance.

Transfer (T to R):

1. Z provides T with input $(\mathbf{transfer}, \mathbf{T}, \mathbf{R}, \mathbf{sid}, \mathbf{cid_n}, \mathbf{cid_{00}}, \mathbf{cid_{01}}, \mathbf{cid_{10}}, \mathbf{cid_{11}}, \mathbf{vid_0}, \mathbf{vid_1})$ (where T is committed to bits b_{00}, b_{01}, b_{10}, b_{11} and R is committed to bits v_0, v_1).

2. T chooses random bits c_0, c_1 and sets

$$\begin{aligned} b'_{00} &= b_{00} \oplus c_0 \qquad & b'_{01} &= b_{01} \oplus c_1 \\ b'_{10} &= b_{10} \oplus c_0 \qquad & b'_{11} &= b_{11} \oplus c_1 \end{aligned}$$

3. T commits to b'_{00}, b'_{10}, b'_{01}, b'_{11}, c_0, c_1 and uses F_{COT} to prove the above relationships.

4. T and R use F_{COT} to transfer b'_{00} and b'_{10} with choice bit v_0.

5. T and R use F_{COT} to transfer b'_{01} and b'_{11} with choice bit v_0.

6. T and R use F_{COT} to transfer c_0 and c_1 with choice bit v_1.

7. R ends up committed to $b'_{v_0 0}$, $b'_{v_0 1}$, and c_{v_1}. R uses F_{COT} to commit to $b_{v_0 v_1}$ and prove that $b_{v_0 v_1} = b'_{v_0 v_1} \oplus c_{v_1}$.

8. Both parties output appropriate receipts.

Transfer (R to T):

Transfers from R to T operate exactly the same way as transfers from T to R.

During a transfer, R learns $b_{v_0 v_1}$ and *only* $b_{v_0 v_1}$. For example, assume that $v_0 = 0$ and $v_1 = 1$. Because $v_0 = 0$, R only learns b'_{00} and b'_{01}. Because $v_1 = 1$, R learns c_1 but not c_0. Therefore, R can determine b_{01} but not b_{00}.

Security for this protocol relies on F_{COT}, which provides ideally secure transfers and proofs. Since T and R interact exclusively through F_{COT}, this makes simulation easy for an ideal adversary S that plays the role of F_{COT}. We therefore state the following theorem without a detailed proof:

Theorem 6.1.1 *The above protocol UC realizes F_{4COT} in the F_{COT}-hybrid model.*

6.1.2 Two-Party Circuit Evaluation

Following the structure used in [30], two-party computation consists of a three-phase circuit evaluation process in the F_{4COT}-hybrid model. Say that parties P_1 and P_2 want to jointly compute an ideal two-party functionality F with security parameter m. F can be represented as a circuit family C_F, where $F_i \in C_F$ is the circuit for F with security parameter i. P_1 and P_2 agree to evaluate F_m through the following (informal) protocol:

Initialization:

Whenever P_1 receives an input bit a [†], P_1 shares it with P_2 by choosing a random bit a_1, setting $a_2 = a_1 \oplus a$, committing to a_1 (using F_{4COT}) and sending a_2 to P_2. P_2 then commits to a_2 and proves that this equals the value received by P_1 (by committing to $a_2' = a_2$ and $a_2'' = 0$, proving $a_2 \oplus a_2' = a_2''$, then opening a_2' and a_2'').

Whenever P_2 receives an input bit b, the same process occurs with roles reversed.

Thus, all inputs x are shared between P_1 and P_2 such that P_1 commits to x_1 and P_2 commits to x_2 where $x_1 \oplus x_2 = x$.

[†]Recall that reactive functionalities allow inputs to become available throughout a computation.

Evaluation:

P_1 and P_2 evaluate F_m on a gate-by-gate basis. Without loss of generality, we assume that each gate is an AND gate or an XOR gate with exactly two inputs and one output. Let gate G compute the relation $c = Y(a, b)$ for $Y \in \{AND, OR\}$, input bits a and b, and output bit c. Evaluation starts with P_1 committed to a_1, b_1 and P_2 committed to a_2, b_2 such that $a_1 \oplus a_2 = a$ and $b_1 \oplus b_2 = b$. Evaluation ends with P_1 committed to c_1 and P_2 committed to c_2 such that $c_1 \oplus c_2 = c$. G is evaluated as follows:

P_1 selects c_1 randomly and commits to it. This fixes $c_2 = c_1 \oplus Y(a_1 \oplus a_2, b_1 \oplus b_2)$. Since P_1 doesn't know a_2 or b_2, P_1 constructs the following table over all possibilities:

$\mathbf{a_2}$	$\mathbf{b_2}$	$\mathbf{c_2}$
0	0	$c_2^{00} = c_1 \oplus Y(a_1, b_1)$
0	1	$c_2^{01} = c_1 \oplus Y(a_1, b_1 \oplus 1)$
1	0	$c_2^{10} = c_1 \oplus Y(a_1 \oplus 1, b_1)$
1	1	$c_2^{11} = c_1 \oplus Y(a_1 \oplus 1, b_1 \oplus 1)$

P_1 then commits to c_2^{00}, c_2^{01}, c_2^{10}, c_2^{11} and uses F_{4COT} to prove that these commitments satisfy the above table (since Y is an AND or XOR relation, this falls within F_{4COT}'s proof capabilities). Finally, P_1 and P_2 conduct a transfer with P_1's inputs as c_2^{00},c_2^{01}, c_2^{10}, c_2^{11} and P_2's inputs as a_2,b_2. This results in P_2 being committed to the appropriate c_2 value.

Output:

Whenever P_1 should produce an output c, P_2 decommits to its share c_2. P_1 then computes $c = c_1 \oplus c_2$ and outputs c.

Whenever P_2 should produce an output, the same process occurs with roles reversed.

We now state the following theorem:

Theorem 6.1.2 *For any two-party ideal functionality F, the above protocol UC realizes F in the F_{4COT}-hybrid model against malicious, adaptive adversaries*

Proof: [30] defines a similar 1-out-of-4 COT ideal functionality known as F^4_{ECOT} and proves that the above protocol UC realizes F in the F^4_{ECOT}-hybrid model. There are exactly two differences between their definition and ours. One, their definition supports proofs on arbitrary binary relations, whereas ours only supports proofs on AND and XOR relations. Two, their definition supports proofs on four bits (i.e. $R(a, b, c) = d$), whereas ours only supports proofs on three bits (i.e. $R(a, b) = c$). These differences only arise in the part of the protocol that proves the evaluation table is correct. Their definition supports proofs for arbitrary gates and requires one proof per table row. Our definition only supports AND and XOR gates and requires multiple proofs per row. But these limitations pose no problems. Restricting ourselves to AND and XOR gates does not reduce the class of computable circuits. Performing multiple proofs instead of a single proof has no security consequences due to the ideal nature of each proof. Therefore, the analysis in [30] remains valid when used with F_{4COT}. $\square$

6.2 Multi-Party Computation

Multi-party computation is almost identical to two-party computation. The main difference is that all operations must be verified by many parties instead of just one. We show how to extend our two-party protocols into the multi-party setting. Parties are assumed to have access to a broadcast channel. This is provided by the ideal functionality F_{BC}. All "real" messages (i.e. messages that are not sent to or from an ideal functionality) are assumed to be sent through F_{BC}. See [12] for a formal definition of F_{BC} and further discussion.

6.2.1 Multi-Party BCX

Given n parties $P_1, ..., P_n$, the multi-party version of F_{BCX} (called F_{mBCX}) is defined as F_{BCX} with the change that commitment, decommitment, and proof receipts are sent to all n parties. In other words, commitments are made to an entire group instead of a single party. F_{mBCX} operates in the (F_{COM},F_{BC})-hybrid model.

We implement F_{mBCX} as follows: P_i commits to bit b by making separate BCX commitments to b for each $P_{j\neq i}$. P_i then proves that all of these commitments are equal by showing that for each $P_{j\neq i}$ and $P_{k\neq j,k\neq i}$, the commitment to P_j equals the commitment to P_k. This is accomplished through a two-party XOR proof, with P_j and P_k collaboratively choosing the random values for protocol steps 9 and 11. In all, the complete commitment requires $O(n^2)$ BCX proofs. [18] illustrates a more efficient method that requires $O(n)$ proofs.

P_i decommits by separately decommitting to each $P_{j\neq i}$. P_i performs an XOR proof by performing a separate XOR proof for each $P_{j\neq i}$.

We claim that this protocol UC realizes F_{mBCX}. The proof of security follows from that of the two-party protocol.

6.2.2 Multi-Party COT

Given n parties $P_1, ..., P_n$, the multi-party version of F_{COT} (called F_{mCOT}) is defined as F_{COT} with the change that commitment, decommitment, proof, and transfer receipts are sent to all n parties (although only the designated receiver learns the value of a transferred bit).

We implement F_{mCOT} by making three changes to the two-party COT protocol. First, parties run in the (F_{mBCX},F_{OT},F_{BC})-hybrid model instead of the (F_{BCX},F_{OT})-hybrid model. Second, when sender P_i and receiver P_j perform a transfer, the code in protocol step 3 is collaboratively chosen by all $P_{k \neq i}$. Third, when P_i and P_j perform a transfer, the subset I_T in step 12 is collaboratively chosen by all $P_{k \neq j}$. These changes guarantee that verification checks are done by all parties instead of only the transfer participants. So even if both the sender and receiver are corrupted, they cannot falsify a transfer without detection. AND proofs are modified in an analogous way.

We claim that this protocol UC realizes F_{mCOT}. The proof of security follows from that of the two-party protocol. We define and assert security for the multi-party version of F_{4COT} (called F_{m4COT}) analogously.

6.2.3 Multi-Party Circuit Evaluation

Given n parties $P_1, ..., P_n$ computing ideal functionality F with security parameter m, F is evaluated as follows:

Initialization:

Whenever P_i receives an input a, P_i shares this input by choosing random values a_j for $j : 1 \leq j \neq i \leq n$, setting $a_i = a \oplus \bigoplus_{j \neq i} a_j$, committing to a_i, and sending each $P_{j \neq i}$ the value a_j. Each P_j then commits to a_j and proves to all parties that this equals the value received by P_i.

Evaluation:

F_m is evaluated on a gate-by-gate basis. Let G be a gate with inputs $a = \bigoplus_{i=1}^{n} a_i$, $b = \bigoplus_{i=1}^{n} b_i$ and output $c = \bigoplus_{i=1}^{n} c_i$. If G is an XOR gate, each party P_i commits to $c_i = a_i \oplus b_i$ and proves this relation. If G is an AND gate, we observe that $(\bigoplus_{i=1}^{n} a_i) \wedge (\bigoplus_{i=1}^{n} b_i) = \bigoplus_{i,j=1}^{n} (a_i \wedge b_j)$. So each pair of parties P_i, P_j conducts a two-party AND evaluation on its input shares (when $i = j$, P_i internally computes $a_i \wedge b_i$ and proves the result is correct). Each P_i then commits to c_i as the XOR sum of all its two-party results and proves that this value is correct.

Output:

Whenever P_i should produce an output c, each $P_{j \neq i}$ decommits to c_j. P_i then computes $c = \bigoplus_{j=1}^{n} c_j$ and outputs c.

We now state the following theorem:

Theorem 6.2.1 *For any multi-party functionality F, the above protocol UC realizes F in the F_{m4COT}-hybrid model against malicious, adaptive adversaries.*

The proof of this theorem follows from the proof for the two-party version. See [30] and [12] for more detailed analysis.

Chapter 7

Conclusion

Let F_{AUTH} be the ideal functionality for authenticated communication (see [7] for the definition and detailed description of F_{AUTH}). We state our final conclusion in the following theorem:

Theorem 7.0.2 *For any multi-party functionality F, there exists a protocol that UC realizes F in the (F_{COM},F_{OT},F_{AUTH})-hybrid model against malicious, adaptive adversaries with no additional assumptions.*

Proof: Theorem 6.2.1 describes a UC protocol for F in the F_{m4COT}-hybrid model. Section 6.2.2 shows that F_{m4COT} can be UC realized in the (F_{mBCX},F_{OT},F_{BC})-hybrid model. Applying corollary 2.2.1 (from the UC composition theorem) produces a UC protocol for F in the (F_{mBCX},F_{OT},F_{BC})-hybrid model. Section 6.2.1 shows that F_{mBCX} can be UC realized in the (F_{COM},F_{BC})-hybrid model. Applying corollary 2.2.1 produces a UC protocol for F in the (F_{COM},F_{OT},F_{BC})-hybrid model. F_{BC} is known to have an implementation that requires no assumptions (see [12]), so applying corollary 2.2.1 produces a UC protocol for F in the (F_{COM},F_{OT})-hybrid model. However, our definition of the UC framework in chapter 2 assumes ideally authenticated communication between parties. This is practically achieved by operating in the F_{AUTH}-hybrid model, so we add F_{AUTH} as a basic primitive. This results in a protocol for F in the (F_{COM},F_{OT},F_{AUTH})-hybrid model. $\square$

Because our protocol operates in the (F_{COM},F_{OT},F_{AUTH})-hybrid model, it uses its bit commitment, oblivious transfer, and authentication primitives as black-boxes by definition. That is, each of these primitives is modeled by an ideal functionality that performs its computations internally and privately from the protocol participants. Parties can use ideal functionalities but cannot "see into" them. This in combination with corollary 2.2.1 guarantees that we can plug in *any* UC implementation for these primitives and retain security. So the assumptions required for our protocol reduce to the assumptions required for UC commitment, UC oblivious transfer, and UC authentication. This makes computation with quantum channels, noisy channels, and other non-standard computational components feasible if we can find appropriate UC commitment, OT, and authentication implementations.

This contrasts with the protocol in [12], which is limited to a standard computational setting. Their protocol uses a "compiler" to convert security against semi-honest adversaries into security against malicious adversaries. Because the compiler requires non-black-box access to the semi-honest protocol, it restricts the class of valid protocols to those that meet the compiler's requirements. Specifically, for semi-honest protocol Π, the compiler requires access to a "next message" function $NM_{\Pi}(x, r, M) = m$ that guarantees a party running Π would produce message m given initial input x, random input r, and message history M. This function is then used with zero-knowledge proofs to make dishonest parties follow the protocol correctly. Such a function is available for all programs based on Turing machines running in a classical computational setting. But it is not necessarily available for quantum programs, programs based on noisy channels, programs where the "source code" is unavailable, or any other programs with fundamental uncertainty in their behavior. It is also not necessarily available for those relativized complexity classes where oracles cannot be modeled as basic Turing machines.

Tangible results for our protocol rely on finding effective UC implementations for bit commitment, OT, and authentication. There has been substantial research in UC commitment in the standard computational setting. [10] formalizes the notion of UC commitment and presents non-interactive commitment schemes based on trapdoor permutations and non-malleable encryption. [12] presents a UC commitment scheme based solely on trapdoor permutations. [23] presents a highly efficient UC commitment scheme with strong binding and hiding properties based on number-theoretic assumptions. UC oblivious transfer has received much less attention. [12] presents an OT protocol based on number-theoretic assumptions that is secure only against semi-honest adversaries. To our knowledge, this is the only known result for OT (although we can use the COT protocol from [30] and decommit to all inputs and outputs to achieve UC OT based on number-theoretic assumptions). For authentication, [7] shows how to achieve UC authentication from public key cryptography and ideal key distribution. [8] develops a more thorough framework that provides unconditionally secure UC authentication given ideal signature and "certification authority" functionalities.

A natural direction for future research is to conduct a definitive study on oblivious transfer in the UC framework. OT is an important and widely applicable primitive that deserves a detailed understanding. There is also substantial opportunity to explore universal composability in non-standard computational settings. [43] has asserted that the UC framework extends naturally into the quantum setting. Results from [22] strongly suggest the feasability of secure two-party and multi-party computation in a quantum UC framework. But to date there are no concrete results in this domain.

Bibliography

[1] Beaver, D. Foundations of secure interactive computing. *Advances in Cryptology - CRYPTO '91* (1991), pp. 377–391.

[2] Beaver, D. How to break a 'secure' oblivious transfer protocol. *Advances in Cryptology - EUROCRYPT '92* (1993), pp. 285–296.

[3] Beaver, D., and Goldwasser, S. Multiparty computations with faulty majority. *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (1989), pp. 468–473.

[4] Beaver, D., Micali, S., and Rogaway, P. The round complexity of secure protocols. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (1990), pp. 25–34.

[5] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988), pp. 1–10.

[6] Bennett, C. H., Brassard, G., Crépeau, C., and Maurer, U. M. Generalized privacy amplification. *IEEE Transactions on Information Theory 41*, 6 (1995), pp. 1915–1923.

[7] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067. Extended abstract in 42nd Annual Symposium on Foundations of Computer Science (2001), pp. 136-145.

[8] Canetti, R. Universally composable signature, certification, and authentication. Cryptology ePrint Archive, Report 2003/239. Extended abstract in Proceedings of the 17th Computer Security Foundations Workshop (2004), pp. 219-235.

[9] Canetti, R. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology 13*, 1 (2000), pp. 143–202.

[10] Canetti, R., and Fischlin, M. Universally composable commitments. Cryptology ePrint Archive, Report 2001/055. Extended abstract in Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology (2001), pp. 19-40.

[11] Canetti, R., Kushilevitz, E., and Lindell, Y. On the limitations of universally composable two-party computation without set-up assumptions. *Advances in Cryptology - EUROCRYPT '03* (2003), pp. 68–86.

[12] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A. Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Report 2002/140, 2002. Extended abstract in 34th Annual ACM Symposium on Theory of Computing (2002), pp. 494-503.

[13] Carter, J. L., and Wegman, M. N. Universal classes of hash functions. *Journal of Computer and System Sciences 18* (1979), 143–154.

[14] Chaor, B., and Kushilevitz, E. A zero-one law for boolean privacy. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing* (1991), pp. 62–72.

[15] Chaum, D., Crépeau, C., and Damgård, I. Multiparty unconditionally secure protocols. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988), pp. 11–19.

[16] Chaum, D., Damgård, I., and van de Graaf, J. Multiparty computations ensuring the privacy of each party's input and correctness of the result. *Advances in Cryptology - CRYPTO '87* (1987), pp. 87–119.

[17] Crépeau, C. Verifiable disclosure of secrets and applications. *Advances in Cryptology - EUROCRYPT '89* (1990), pp. 181–191.

[18] Crépeau, C., van de Graaf, J., and Tapp, A. Committed oblivious transfer and private multi-party computation. *Advances in Cryptology - CRYPTO '95* (1995), pp. 110–123.

[19] Crescenzo, G. D., Ishai, Y., and Ostrovsky, R. Non-interactive and non-malleable commitment. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing* (1998), pp. 141–150.

[20] Crescenzo, G. D., Katz, J., Ostrovsky, R., and Smith, A. Efficient and non-interactive non-malleable commitment. Cryptology ePrint Archive, Report 2001/032, 2001.

[21] Damgård, I. Efficient concurrent zero-knowledge in the auxiliary string model. *Advances in Cryptology - EUROCRYPT '00* (2000), pp. 418–430.

[22] Damgård, I., Fehr, S., and Salvail, L. Zero-knowledge proofs and string commitments withstanding quantum attacks. *Advances in Cryptology - CRYPTO '04* (2004), pp. 254–272.

[23] Damgård, I., and Nielsen, J. B. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology* (2002), pp. 581–596.

[24] Damgård, I., and Nielsen, J. B. Universally composable efficient multiparty computation from threshold homomorphic encryption. *Advances in Cryptology - CRYPTO '03* (2003).

[25] Dolev, D., Dwork, C., and Naor, M. Non-malleable cryptography (preliminary version). *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), pp. 542–552. Final version in SIAM Journal of Computing, Vol. 30, No. 2 (2000), pp. 391-437.

[26] Dwork, C., Naor, M., and Sahi, A. Concurrent zero-knowledge. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing* (1998), pp. 409–418.

[27] Fischlin, M., and Fischlin, R. Efficient non-malleable commitment schemes. *Advances in Cryptology - CRYPTO '00* (2000), pp. 413–431.

[28] Galil, Z., Haber, S., and Yung, M. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. *Advances in Cryptology - CRYPTO '87* (1987), pp. 135–155.

[29] Garay, J., and MacKenzie, P. Concurrent oblivious transfer. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science* (2000), pp. 314–324.

[30] Garay, J. A., MacKenzie, P., and Yang, K. Efficient and universally composable committed oblivious transfer and applications. *Proceedings of the First Theory of Cryptography Conference* (2004), pp. 297–316.

[31] Goldreich, O. *Foundations of Cryptography*, vol. 1. Cambridge University Press, 2001.

[32] Goldreich, O. Concurrent zero-knowledge with timing, revisited. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing* (2002), pp. 332–340.

[33] Goldreich, O., and Kahan, A. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology* (1996), pp. 167–190.

[34] Goldreich, O., and Krawczyk, H. On the composition of zero-knowledge proof systems. *SIAM Journal of Computing 25*, 1 (1996), pp. 169–192.

[35] Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game. *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (1987), pp. 218–229.

[36] Goldreich, O., and Oren, Y. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology 7*, 1 (1994), pp. 1–32.

[37] Goldwasser, S. Multi party computations: Past and present. *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing* (1997), pp. 1–6.

[38] Goldwasser, S., and Levin, L. Fair computation of general functions in presence of immoral majority. *Advances in Cryptology - CRYPTO '90* (1990), pp. 75–84.

[39] Goldwasser, S., and Micali, S. Probabilistic encryption. *Journal of Computer and Systems Sciences 28*, 2 (1984), pp. 270–299.

[40] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof systems. *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (1985), pp. 291–304.

[41] Kilian, J. A note on efficient zero-knowledge proofs and arguments. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing* (1992), pp. 723–732.

[42] Kilian, J., and Petrank, E. Concurrent and resettable zero-knowledge in poly-logarithmic rounds. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing* (2001), pp. 560–569.

[43] Mayers, D. Quantum universal composability. Mathematical Sciences Research Institute: Workshop in Quantum Information and Cryptography, 2002.

[44] Micali, S., and Rogaway, P. Secure computation. Unpublished manuscript, 1991. Abstract in CRYPTO '91 (1991), pp. 392-404.

[45] Naor, M., and Reingold, O. Public-key cryptosystems provably secure against chosen ciphertext attack. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (1990), pp. 427–437.

[46] Rackoff, C., and Simon, D. Non-interactive zero-knowledge proofs of knowledge and chosen ciphertext attack. *Advances in Cryptology - CRYPTO '91* (1992), pp. 433–444.

[47] Richardson, R., and Kilian, J. On the concurrent composition of zero-knowledge proofs. *Advances in Cryptology - EUROCRYPT '99* (1999), pp. 415–431.

[48] Sahai, A. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science* (1999), pp. 543–553.

[49] Sudan, M. Coding theory: Tutorial and survey. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science* (2001), pp. 36–53.

[50] Yao, A. Protocols for secure computation. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (1982), pp. 160–164.

[51] Yao, A. How to generate and exchange secrets. *Proceedings of the 27th Annual Symposium on Foundations of Computer Science* (1986), pp. 162–167.