ABSTRACT

Title of dissertation: ON FAIRNESS IN SECURE COMPUTATION

S. Dov Gordon, Doctor of Philosophy, 2010

Dissertation directed by: Professor Jonathan Katz
Department of Computer Science and UMIACS

Secure computation is a fundamental problem in modern cryptography in which multiple parties join to compute a function of their private inputs without revealing anything beyond the output of the function. A series of very strong results in the 1980's demonstrated that any polynomial-time function can be computed while guaranteeing essentially every desired security property. The only exception is the *fairness* property, which states that no player should receive their output from the computation unless all players receive their output. While it was shown that fairness can be achieved whenever a majority of players are honest, it was also shown that fairness is impossible to achieve in general when half or more of the players are dishonest. Indeed, it was proven that even boolean XOR cannot be computed fairly by two parties

The fairness property is both natural and important, and as such it was one of the first questions addressed in modern cryptography (in the context of signature exchange). One contribution of this thesis is to survey the many approaches that have been used to guarantee different notions of *partial* fairness. We then revisit the topic of fairness within a modern security framework for secure computation. We demonstrate that, despite the strong impossibility result mentioned above, certain interesting functions can be computed fairly, even when half (or more) of the parties are malicious. We also provide a new notion of partial fairness, demonstrate feasibility of achieving this notion for a large class of functions, and show impossibility for certain functions outside this class. We consider fairness in the presence of *rational* adversaries, and, finally, we further study the difficulty of achieving fairness by exploring how much external help is necessary for enabling fair secure computation.

ON FAIRNESS IN SECURE COMPUTATION

by

Samuel Dov Gordon

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2010

Advisory Committee:
Professor Jonathan Katz
Professor William Gasarch
Professor David Mount
Professor Bobby Bhattacharjee
Professor Lawrence Washington

© Copyright by
Samuel Dov Gordon
2010

Acknowledgments

I began graduate school knowing almost nothing about cryptography. I asked Jonathan Katz to advise me primarily because he seemed excited and eager to teach. He has taught me a tremendous amount about cryptography in the last four years, and always with complete patience and interest. He almost always found time for me when I needed it, even when it was in short supply, and no matter my mood when I went into his office, I left feeling motivated and excited.

Jonathan also kept my eye on a bigger picture, teaching me to write papers, research grants and talks, and providing me with many opportunities to pursue exciting collaborations and learning experiences. I entered graduate school with an idealistic view of academic life. It is easy to become discouraged as a more realistic understanding sets in, and I credit Jonathan for teaching me to navigate not only cryptographic research, but academic life in general. The most important aspect of the graduate experience is the student / advisor relationship. I am very lucky to have had Jonathan as an advisor. He has been a model teacher.

I want to thank Arkady Yerukhimovich for six years of friendship and collaboration. We began graduate school with similar interests, and we have worked closely together the entire time. He has always been available to help me think through research problems, and to provide feedback and insight, no matter the topic. Working and learning with him has made these years a lot of fun. I want to thank all of Jonathan's past and present students as well, and Ranjit Kumaresan in particular, for his continued enthusiasm, his friendship, and for the many hours he has spent with me at the white board.

I was fortunate to form many collaborations over the last four years. The research in this thesis began primarily out of a collaboration with Jonathan Katz, Carmit Hazay, and Yehuda Lindell. My other co-authors include Tal Moran, Yuval Ishai, Amit Sahai and Rafail Ostrovsky. I was lucky to have been hosted by Moni Naor at Weizmann for a summer, where I worked with Moni, Tal Moran, Gil Segev, Noam Livne, and Eran Omri. I also had a wonderful summer at IBM, where I worked with Tal Rabin, Vinod Vaikuntanathan, and Hugo Krawczyk; their research group is among the warmest, and I enjoyed many fun hours there with Jonathan, Tal, Vinod, Hugo, Shai, Rosario, Muthu, Craig and Charanjit. I have met and learned from many more people at conferences, workshops and visits. Our research community provides a warm and encouraging environment for research, and I appreciate everyone that contributes to this atmosphere. I am again grateful to Jonathan for facilitating all of these experiences.

Finally, I may never have considered this career path without a family that so fully values learning and education. My own excitements and frustrations were well understood and completely shared by my parents. This work is dedicated to them.

Contents

1 Introduction **6**
- 1.1 Contributions . . . 8
- 1.2 A Survey of Fairness in Secure Computation . . . 9

2 Definitions and Preliminaries **23**
- 2.1 Basic Notation . . . 23
- 2.2 Basic Cryptographic Primitives . . . 24
- 2.3 Secure Two-Party Computation with Complete Fairness . . . 26
- 2.4 Secure Two-Party Computation With Abort . . . 28
- 2.5 Secure Multi-Party Computation with Complete Fairness . . . 29
- 2.6 Secure Multi-Party Computation With Designated Abort . . . 30
- 2.7 The Hybrid Model . . . 31
- 2.8 A Canonical Form for Fair Two-Party Computation . . . 32

3 Complete Fairness in Secure Two-Party Computation **35**
- 3.1 Fair Computation of the Millionaires' Problem (and More) . . . 35
  - 3.1.1 The Protocol . . . 36
- 3.2 Fair Computation of Functions with an Embedded XOR . . . 43
  - 3.2.1 Proof of Security for a Particular Function . . . 46
  - 3.2.2 Extending the Protocol to Other Functions . . . 56
  - 3.2.3 A Characterization of When Protocol Π_{EXOR} is Secure . . . 58
- 3.3 A Lower Bound for Functions Containing an Embedded XOR . . . 63
  - 3.3.1 Preliminaries . . . 63
  - 3.3.2 The Proof . . . 66

4 Complete Fairness in Secure Multi-Party Computation **71**
- 4.1 Fair Computation of Majority for Three Players . . . 71
- 4.2 A Lower Bound on the Round Complexity of Majority . . . 81
  - 4.2.1 Proof Overview . . . 82
  - 4.2.2 Proof Details . . . 83
- 4.3 Fair Computation of OR for n Players . . . 86

5 Partial Fairness in Secure Two-Party Computation **90**
- 5.1 $\frac{1}{p}$-Secure Computation of General Functionalities . . . 91
  - 5.1.1 A Useful Lemma . . . 92
  - 5.1.2 $\frac{1}{p}$-Security for Functionalities with Polynomial-Size Domain . . . 93
  - 5.1.3 $\frac{1}{p}$-Security for Functionalities with Polynomial-Size Range . . . 97
- 5.2 Optimality of Our Results . . . 100

5.2.1 Impossibility of $\frac{1}{p}$-Security and Security-with-Abort Simultaneously . . 100
5.2.2 Impossibility of $\frac{1}{p}$-Security for General Functions 102

6 Fairness When Players are Rational **104**
6.1 Definitions from Game Theory 106
6.2 A Protocol for Rational Secret Sharing 107
6.3 Discussion 110

7 Fair Primitives for Secure Computation **113**
7.1 A Complete Primitive for Fair Two-Party Computation 113
7.2 A Lower Bound on the Size of Complete Primitives 117

Chapter 1

Introduction

In 1980, two years after Rivest, Shamir, and Adleman [82] introduced the first digital signature scheme, Shimon Even and Yacov Yacobi published a paper titled *Relations Among Public Key Signature Schemes* [34]. In this work, they explored the power of digital signatures, proving, among other results, that it is impossible for two distrusting parties to *fairly* exchange signatures on a contract. Their intuition was simple and interesting: as the two parties exchange messages back and forth, there must exist some point in the protocol at which one party has enough information to produce his opponent's signature, while the other party does not. The party that receives a signature first can terminate the protocol at that point, and violate the fairness of the exchange. It seems an obvious point, but it sparked a very interesting line of research.

In the summer of 1980, Manuel Blum suggested to Even a way to circumvent this impossibility result. While it is true that there exists a point in the protocol at which only one player can recover the other's signature, it may be that they can do this only at great computational cost. Blum's idea (roughly) was to have the players begin by exchanging an *encryption* of the signatures.[1] Technically, it is still true that the first player to receive the encryption has an unfair advantage, as they can run away with the ciphertext and recover the output through "brute-force". But this is computationally infeasible, and should not be considered unfair. Once the players have encryptions of the output, they alternate sending one bit of the decryption key, along with a proof of correctness, until the entire key has been sent. Of course, the leading player has a continuing advantage over the other: informally, at any point in the protocol it is twice as easy for him to recover the output as it is for his opponent. But on an intuitive level, the solution seems fair. Soon after their discussion, Blum [15] and Even [32] each wrote papers that made use of this very idea.

So who was right? Even and Yacobi [34], who proved that signature exchange was impossible, or Blum [15] and Even [32] who demonstrated the first protocols for fair exchange? Of course, there is no contradiction — like much of modern cryptography, the subtleties arise in the definitions, and in this case, the authors were (implicitly) considering two different notions of fairness. Although the impossibility result was informal, the intuition can easily be used to formally rule out *completely fair* protocols for signature exchange. In contrast, the positive results allow one player a slight advantage, and therefore do not violate such a proof. We will return to discuss the question of how fairness should be defined, and we will reconsider the above results at that time. Before doing that, we turn briefly to consider a separate

[1] Actually, Blum doesn't use encryption, but rather demonstrates how two players can exchange signing keys bit by bit. See Section 1.2 for details.

line of research that would merge with fairness only in 1986.

Secure computation

> "Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation?" [88]

This was the question that opened Andrew Yao's groundbreaking 1982 paper on secure computation [88]. While the image of the millionaires is certainly compelling, this particular problem is only one example of several similar questions already considered by others, such as how to play poker without cards, or vote without ballot boxes. What made Yao's paper stand out was his presentation of a "unified view of secure computation." He generalized the above problems, unifying them with one single question: Alice and Bob hold private inputs, x and y (respectively), and wish to compute $F(x, y) = (f^1(x, y), f^2(x, y))$, where Alice receives output $f^1(x, y)$ and Bob receives $f^2(x, y)$. How can they do this without revealing anything more than their intended outputs? In addition to presenting the general question, which encompasses all of the problems previously considered in the field, Yao gave a general solution, demonstrating that *any* polynomial-time function of this form can be computed, while guaranteeing the privacy of the inputs.

Then, in a second paper that was published in 1986, Yao merged the questions of fairness and secure computation, showing that any two-party secure computation can be computed fairly. Here, Yao defined a notion of fairness similar to the one intended by Blum: his protocol ensures that no player has a significant computational advantage in recovering their output before the other [89]. In the remainder of that decade there was a flurry of impressive works on secure computation [45, 37, 22, 13, 22, 11, 47],[2] demonstrating very strong feasibility results. However, the definitions of security varied from paper to paper, and in some cases they were hard to understand and equally hard to use (in the sense that proofs of security were cumbersome and often were not presented). Most of the definitions failed to capture all possible attacks (though it should be noted that only the definitions of security were flawed, and not the protocols themselves). Finally, in the Crypto 1991 conference, there were two (similar) papers that unified and strengthened the varying security definitions: one by Micali and Rogaway [73] and another by Beaver [10]. The definition of security that emerged as the accepted definition, which is presented in Section 2.3 (Definition 2.3.1), is based on the definitions presented in these two papers, and on the works of Canetti [20] and Goldreich [42]. Collectively, after a decade of research, these works resulted in a single, beautiful framework for defining and proving security in interactive computation. Furthermore, they provided strong feasibility results enabling the secure computation of any polynomial-time function, and even guaranteeing complete fairness whenever a strict majority of players are honest.

Fairness in Secure Two-Party Computation: Unfortunately, when we revisit the impossibility result of Even and Yacobi [34] in the context of this framework, we find that it still applies. Their work rules out completely fair protocols for two-party signature exchange, and our preferred security definition (Definition 2.3.1) requires complete fairness in the protocol. More technically, although it was never stated explicitly, the assumption made by Even and Yacobi was that the protocol must always specify well defined output for the honest player, even when the dishonest player aborts. The same assumption is also implicitly demanded by Definition 2.3.1. Under this assumption, at some point during a fair signature exchange,

[2] There exist several excellent surveys of secure computation [42, 27], so we do not review these works here.

the protocol must specify *correct* output (i.e., a valid signature) for one player while not yet specifying it for the other, and in this sense any protocol must be unfair.[3] Furthermore, in 1986, Richard Cleve proved that two players cannot even agree on an unbiased random bit [25], assuming, again, that the players always have well defined output. His result implies that it is impossible for two parties to fairly exchange two bits while satisfying Definition 2.3.1 (since the XOR of two random bits yields an unbiased coin flip).

As we will soon see, there is a rich body of work achieving various notions of partial fairness. We have already briefly described one approach suggested by Blum in 1980. However, most of this work appeared before the preferred security definition had emerged. Now that we have adopted this notion of security, some very interesting questions about fairness resurface. We address the following questions in this thesis.

1.1 Contributions

> What functions *can* be computed with complete fairness, without an honest majority, satisfying Definition 2.3.1?

As mentioned above, we already know we cannot compute boolean XOR without a strict honest majority. This is most likely why nobody asked this question after the definition of secure computation first appeared in 1991. Surprisingly, we show that certain interesting functions *can* be computed in this setting, including Yao's millionaire problem.[4] These results appear in Chapters 3 and 4.

> What alternative security definitions can we use that will allow some meaningful notion of *partial fairness*?

The works on fair exchange clearly achieve some notion of partial fairness, even if the protocols do not satisfy the strong definition that we would prefer. But there are very compelling reasons for sticking to the general framework presented in Chapter 2. Recall that Definition 2.3.1 guarantees complete security, including fairness, while Definition 2.4.1 guarantees complete security excluding fairness. In Chapter 5 we present a new definition in the same framework that allows for a meaningful middle ground.

> Fairness is impossible to achieve in general when half (or more) of the players are malicious. Can we do better by considering weaker adversaries?

Our discussion until now has revolved around malicious adversaries that may act in a worst-case manner to ruin our protocols. Most cryptographic tasks can be achieved even in this setting, which is why we typically design protocols with such adversaries in mind. However, fairness *cannot* in general be achieved in this setting, so this may be one case where it pays to consider weaker adversaries. In particular, we consider achieving fairness when the players are *rational* in the standard game-theoretic sense. Our work in this direction is presented in Chapter 6.

[3] It is interesting to think about how the protocols of Blum [15] and Even [32] violate this assumption. Notice that the protocol cannot specify whether one player should choose to recover the output through brute-force if the other aborts in the middle of the protocol. If it *did* specify this, the aborting player can always choose to abort one round earlier and violate fairness. It follows that the player's output is not well defined when the other player aborts.

[4] Recall that Yao showed how to compute any two-party function fairly, but only according to a notion of partial fairness we call gradual release.

Fairness is easily achieved with the help of a third party. How much help is necessary?

In Chapter 7 we present a work that quantifies *how much* help is necessary for achieving fairness. Ideally, we would like to use a third party that is stateless, oblivious even to the particular function being computed. We demonstrate that this is possible. We also quantify the help provided by the size of the input given to the third party, and we give upper and lower bounds on feasibility.

1.2 A Survey of Fairness in Secure Computation

There is a long history of work on the topic of fair exchange, most of it predating the now-standard definitions of security that are presented in Chapter 2. Consequently, each work introduced its own notion of security, or simply did not define security at all, making it relatively hard to formally state what was achieved. Nevertheless, it is worth surveying the techniques used and, when possible, the notions of security that were considered. We try to do that here. We have divided the works mainly by the definition of fairness they achieve, hoping to make it easier to read.

Impossibility of Complete Fairness

Even and Yacobi, 1980: Even and Yacobi [34] introduced the question of fair exchange just a few years after digital signature schemes had been introduced [29, 82, 78]. The authors were interested in exploring the possible applications of digital signatures, and their relationship to other cryptographic primitives. Specifically, they study the relationship between signature schemes and identification schemes, the round complexity of signature schemes, and, finally, the application of exchanging signatures (which they call a "public key agreement scheme"). As described in the previous section, they prove impossibility of the latter through a simple informal argument, claiming that one party must have enough information to efficiently produce a verifiable signature before the other party can do the same. As we will see below, this is not technically true (if we apply the standard asymptotic definition of "efficient"), but the intuition is correct. Under the assumption that both players always have well defined output, even when their opponent aborts, we can prove that no protocol offers completely fair signature exchange.

Cleve, 1986: Under the same assumption that players always have well defined output, Cleve strengthens the earlier impossibility result [25], proving that even completely fair bit exchange is impossible. Actually, framing his result in this light skews his motivation, which stemmed from two recent results on distributed coin flipping [7, 18], and not from recent work on fair exchange. Technically, his theorem states that there does not exist an efficient protocol enabling two players to agree on an unbiased coin. Nevertheless, his result has the implications suggested above: if two players could fairly exchange two bits, they could choose random bits, exchange them, and output their XOR to produce an unbiased coin. Even though this was not his motivation, he certainly recognized the implication. Indeed, Beaver and Goldwasser's 1989 paper [11] (discussed below) will cite Cleve's result as proof that "perfect fairness is not achievable" for general secure computation, and, citing a personal communication with Cleve, they state that:

> If p and q are the maximal a priori probabilities of the players to know the value of $f(x_1, x_2)$, given x_1 or x_2 respectively, then for any two-party protocol running for κ rounds there exists a quitting strategy of one player enabling him to predict f with probability at least $\frac{\min(1-p,1-q)}{2\kappa}$ better than the other player.

It is likely that Cleve's result is what prevented people from asking whether anything of interest could be computed with complete fairness; the results we present in Chapters 3 and 4 are surprising in light of this earlier work.

Cleve's impossibility result is remarkably simple to read and understand. For any κ round protocol, he demonstrates that there exist 4κ different adversaries such that at least one adversary inflicts a polynomial bias on the outcome of the coin. The adversaries are easily described: $\mathcal{A}_b^{(i)}$ runs honestly for i rounds, and then tests internally what his own output would now be if the honest player were to abort. (Here we are using the assumption described earlier that both players always have well defined output in case the other player aborts.) If his output is b, he continues honestly for one more round and then aborts; if it is $1-b$, he aborts immediately. The intuition for why one of these adversaries is successful is as follows. Take any coin-flipping protocol, and consider an execution that results in output b. At the very start of the protocol, $\Pr[\text{OUT} = b] = \frac{1}{2}$, while at the end of the protocol, $\Pr[\text{OUT} = b] = 1$; since there are only a polynomial number of rounds, somewhere in the middle of the protocol, there must be a "gap" where this probability makes a polynomial jump towards 1. Cleve proves very cleanly that this gap exists, and demonstrates that one of these adversaries exploits that gap and biases the outcome.

Gradual Release

Despite the above impossibility results, as we have already discussed, it is possible to achieve various notions of partial fairness. *Gradual release* is the technique used in most of the feasibility results that we present below. As we have already briefly described, the idea used in all of these works is that in each round of interaction, it becomes progressively easier (cryptographically) to recover the output.

Blum, 1983: Although Blum's 1983 work [15] was not the first result using gradual release, it is cited in nearly every work on fair exchange after Even and Yacobi's 1980 result [34]. It seems that this paper sparked the long line of feasibility results.[5] Blum provided a way for two players to fairly exchange secret keys using gradual release. He then suggests several applications that this might be used for, including signature exchange. In his work, the two public keys are composite numbers of the form $N_A = p_A \cdot q_A$, and $N_B = p_B \cdot q_B$, where p_A, q_A, p_B, q_B are all large primes, and the users wish to exchange (p_A, q_A) and (p_B, q_B). The protocol relies on the following facts from number theory. Any a relatively prime to N_A has 4 square roots mod N_A, and knowing its roots (for any such a) is sufficient for efficiently factoring N_A. (Actually, only two square roots are necessary, if they are they are paired appropriately, but four certainly suffices.) Furthermore, given p_A and q_A, it is easy to find square roots mod N_A.

The protocol proceeds as follows. The player that wishes to factor N_A chooses a random element $a \in Z_{N_A}$ and sends $a^2 \bmod N_A$ to his opponent. The player learning the factors of N_B acts similarly sending some $b^2 \bmod N_B$. Note that each player knows two roots of the number they sent: $\pm a$ for the first player, and $\pm b$ for the second. Furthermore, they can each compute

[5] There are also several later references to a 1981 technical report by Blum titled "Three Applications of the Oblivious Transfer", which we could not find. This title refers to Rabin's work [79] which we describe below.

the four square roots of the number they received. They then alternate sending messages for i rounds, where in round i, each player sends the ith bit of all four square roots. Because neither player knows which two roots the other holds, they cannot send incorrect bits without getting caught! This suffices for soundness $1/2$: to further reduce the probability of cheating, the players instead start by each sending κ squares, and then alternate sending one bit of all 4κ roots in each round. At Crypto in 1983 and 1984, Tedrick [85, 86] demonstrated how to slow down the advantage of the leading player in Blum's protocol. Instead of sending a bit of the secret key, the user can cut the search space by a lesser amount by sending a statement like "the last 3 bits are not 001".

Unfortunately, Blum's result relied on several strong assumptions (all very precisely stated in his paper). One of these assumptions was proven false by Håsted and Shamir a few years after his paper appeared [54]. Blum had assumed that having a few bits from *many* square roots is no more useful (with respect to factoring N_A or N_B) than having the same number of bits from only a single square root. Put more formally, given the k most (or least) significant bits from the roots of $y_1 = a_1^2, \ldots, y_\kappa = a_\kappa^2$, he assumed that it is no easier to find the remaining bits of any root a_i than it would be if given only k bits of a_i. Håstad and Shamir demonstrate how to factor in polynomial time given only $O(\log N_A/\kappa)$ bits of each of the $O(\kappa)$ unknown variables. As κ grows, then, we require fewer and fewer bits before we can factor.

Even, 1981: This work by Even [32] was done concurrently with Blum's work on secret exchange. Even refers to his earlier impossibility result saying "it was shown by Even and Yacobi [34] that no deterministic protocol exists [for fairly committing two players to a contract]... During the summer of 1980, in a conversation, M. Blum suggested the use of randomization for such protocols." It is not clear why Even focuses on whether the protocol is randomized, since the prior impossibility result rules out even randomized protocols, as long as they meet the assumption described above. (We cannot easily say anything about the impossibility result for protocols that do not meet this assumption.) Nevertheless, regardless of how he reconciled it with the prior impossibility result, he gives the first published work on gradual release (chronologically).

Even's result relies on *Merkle puzzles* [71], which are (informally) defined as follows. Given $(M, F_k(M))$ where M is chosen at random and F_k is a keyed one way function, the puzzle is to find the key k. It is assumed that given t bits of k, the only way to find k is to try all $|k| - t$ bits[6]; this allows the issuer of the puzzle to control its exact difficulty by publicizing an appropriate fraction of k. Even assumes that the value of the contract to both players is some fixed value V, and defines the contract to be valid only if they have each signed it $\kappa + 1$ times (where κ, here and throughout, is a security parameter). His protocol proceeds by having both players encrypt 2κ signatures on the contract using encryption keys $k_1, \ldots, k_{2\kappa}$, and then generate a Merkle puzzle for each encryption key, where each puzzle has difficulty $2V/\kappa$. The players exchange the encrypted signatures and the Merkle puzzles: at this point, either player could brute-force $\kappa + 1$ puzzles at a cost of more than $2V$, but by assumption, neither player would care to spend the computational resources. Now the players take turns sending encryption keys, one at a time; each player chooses at random which key he would next like to receive. If a player ever aborts, they only have a small computational advantage ($2V/\kappa$) in recovering the other's signature. Because there are 2κ encrypted signatures, if a player cheats by forming bad puzzles, they will either be caught with high probability (if many of them are bad), or it will not greatly hinder the signature reconstruction (if only a few are bad). This provides a tradeoff: if a player cheats, then either we catch him with high probability, or his

[6]This is a worrisome assumption, which we discuss more below.

cheating will not significantly slow us down when we try to brute-force a signature.

Goldreich, 1983: Goldreich published a short note at Crypto 1983 [41] that simplifies Even's scheme [32]. He instructs the two parties to each generate κ Merkle puzzles, and then sign a statement saying they are committed to the contract if the other can either solve all κ puzzles, *or prove that they're not all solvable*. They then exchange the solutions, one at a time. The point is that it is not necessary to catch a party that cheats: instead we can define a signature on a bad puzzle as a commitment to the contract.

Rabin, 1981: This early work on fairness [79] does not use the technique of gradual release. In fact, although its subject is the fair exchange of two bits, it is not typically remembered for dealing with the topic of fairness at all. Rabin's seminal paper introduced a new tool called oblivious transfer (OT), which has since become an essential cryptographic primitive. It would later be proven both necessary and sufficient for performing general secure computation (without fairness) [61, 56]. However, even before that, OT would be used in several works on gradual release, which is why we present the result here.

Oblivious transfer, as defined by Rabin, is a protocol involving two players, a sender and a receiver. The sender holds input $N = p \cdot q$, where p and q are large primes, and at the end of the protocol, with probability $1/2$ the receiver learns p and q and with the remaining probability he learns nothing; the sender remains oblivious as to whether the receiver learns p and q. Rabin constructs this scheme using the same number theoretic properties that appeared in Blum's protocol. The receiver begins by choosing a value a and sending $a^2 \bmod N$ to the sender. The sender responds with one of the square-roots of a^2. If he happens to respond with $\hat{a} = \pm a$, then the receiver learns nothing. However, if he responds with either of the other two roots, the receiver can easily recover p and q.

The definition of fairness that Rabin considers for bit exchange (left implicit) is that neither player should have complete confidence in their output bit before the other. He assumes that the players are honest, except that they may abort once they have complete confidence in the answer. (In order to enforce honest behavior, he suggests signing all messages so that they can be shown to a judge later. However, he is not willing to rely on the judge for the fairness property.) He also assumes that if either player learns their output, the other player will know that they learned it. This is a strange assumption, but can perhaps be motivated when the outcome of the protocol leads to an observable action in the real world. Finally, he allows the protocol to have a polynomial probability of failure, in which case it terminates with neither party learning the bit (and cannot be restarted).

This is a long list of assumptions, which certainly seem to weaken the importance of the result, but the technique that he uses is very interesting. It turns out to be very similar to the one we use in Chapter 3 to compute Yao's millionaire problem with complete fairness and, surprisingly, it is unlike any other approach used in the 30 years between those two works. The key idea is to design the protocol such that the very act of aborting reveals some crucial information. To exchange two input bits, b_A and b_B, Alice and Bob each begin by choosing (and announcing) a public key, N_A and N_B respectively. They each take a turn playing the role of the sender in an execution of an OT. We let b_A^{OT} denote the outcome of the execution when Alice is the receiver, namely $b_A^{\mathsf{OT}} = 1$ if she learned the factors of N_B, and $b_A^{\mathsf{OT}} = 0$ otherwise. We define b_B^{OT} analogously. Then, Alice sends $b_A \oplus b_A^{\mathsf{OT}}$ and Bob sends $b_B \oplus b_B^{\mathsf{OT}}$. Finally, Alice sends an encryption of her input bit using her public key N_A (without knowing whether Bob has learned its factors as a result of the OT), and Bob responds in kind. If Bob neglects to send his final message, and instead goes to use his learned bit, Alice will know that he learned the factors of N_A; she will deduce that the value of $b_B^{\mathsf{OT}} = 1$, and thus recover

Bob's input bit, reinstating fairness. Similarly, if Bob sends his encryption and Alice recovers b_B, Bob will know when she uses the bit that the value of $b_A^{\mathsf{OT}} = 1$, and will thus recover b_A. With probability $3/4$, the protocol ends with both players recovering the other's input bit in the above manner. With the remaining probability, neither player learns anything.[7] The motivation is forced, and this is hardly a satisfying protocol for bit exchange, but the techniques are very interesting.

Even, Goldreich and Lempel, 1982: In Crypto 1982, Even et al. [33] published another paper that made use of the gradual release technique. (A more complete version with a slightly different protocol would appear in Communications of the ACM in 1985.) It is essentially a hybrid of the works by Blum [15] and Even [32] described above, through the use of Rabin's new oblivious transfer primitive. In the 1982 version, they present a new, more general construction of OT from a public key encryption scheme (of a particular form), where the value learned by the receiver (with probability half) is some message M rather than the prime factors of a public key.

Similar to Even's approach, they define a contract to be signed if the party can produce the solutions to κ Merkle puzzles. Similar to Blum's approach, they catch a cheating player by using OT to send (an expected) half of the solutions up front. The players then alternate sending one bit of every solution. A cheater is caught if they send the wrong bit in a solution that was already learned through OT.

In the version that appeared in 1985, they define and implement a 1-out-of-2 OT, which is the notion used today in reference to OT. In this cryptographic primitive, the sender has two values, rather than one, and the receiver always learns exactly one value. The sender is oblivious to which value was learned. The players begin the protocol with κ pairs of unsolved Merkle puzzles, and they define a signature to be a pair of solutions to any pair of puzzles. The players first receive exactly one solution to each pair through OT, and then exchange the bits of all 2κ solutions, one at a time. They use the solution they learned through OT in order to prevent the sender from cheating as they send the bits of the solutions. This approach removes any advantage that might arise from statistical deviation in the number of puzzle solutions received through OT in the 1982 version of the protocol.

Yao, 1986: As we mention in the previous section, Yao's 1986 paper played a crucial step in developing the formal study of fairness [89]. Just as he was the first to generalize the notion of secure computation in his 1982 paper [88], this work is the first to generalize fairness. Whereas previous work studied a few select problems, like signature exchange, certified mail and bit exchange, Yao extended the notion of fairness, arguing that it should be a property of any secure computation. If Alice and Bob are computing some function $F(x, y)$, we should guarantee that neither player receives output before the other.

Yao gives a formal definition of three security properties: validity, privacy, and fairness. We focus here only on the last property, for which he requires that if one player can recover the output, the other player should have some "recovery" protocol, $\mathcal{R}$, that allows him to do the same. The specification of $\mathcal{R}$ may be dependent on the adversarial code $\mathcal{A}$. It takes as input the honest player's input and the honest player's view in the protocol execution. With significant probability, it outputs the correct output whenever the adversary recovers the correct output. Formally (but modified to make the exposition clearer):

[7] It seems Rabin could have further assumed that players always use their output within some known time period. Then the players can each deduce their output from the fact that the other player did *not* learn anything. The protocol would guarantee output in this case.

Definition 1.2.1 *A protocol π constitutes a fair protocol for computing $\mathcal{F} = \left(f_\kappa^1, f_\kappa^2\right)$ if for any fixed constant c, any probabilistic, polynomial time adversary $\mathcal{A}$, there exists a probabilistic, polynomial time recovery protocol $\mathcal{R}$ such that the following holds for any probabilistic, polynomial time algorithm $\mathcal{G}$:*

$$\begin{aligned}\Pr\left[\left(\text{OUT}_\pi^{\mathcal{A}} = f_\kappa^2(x,y)\right) \wedge \left(\mathcal{R}(1^\kappa, x, \text{VIEW}_\pi^{\mathcal{H}}) \neq f_\kappa^1(x,y)\right)\right] \\ \leq \Pr\left[\mathcal{G}(1^\kappa, y) = f_\kappa^2(x,y)\right] + O(\kappa^{-c})\end{aligned}$$

where $\text{OUT}_\pi^{\mathcal{A}} = \text{OUT}_\pi^{\mathcal{A}}(1^\kappa, x, y; r_{\mathcal{A}}, r_{\mathcal{H}})$ is the output of the adversary when interacting with $\mathcal{H}$, each having inputs $(1^\kappa, y)$ and $(1^\kappa, x)$ (respectively) and random coins $r_{\mathcal{A}}$ and $r_{\mathcal{H}}$ (respectively), and $\text{VIEW}_\pi^{\mathcal{H}} = \text{VIEW}_\pi^{\mathcal{H}}(1^\kappa, x, y; r_{\mathcal{A}}, r_{\mathcal{H}})$ is the view of the honest player in the same interaction.

The role of $\mathcal{G}$ in the above definition is to capture the fact that the adversary may be able to guess the output just by looking at his input. We note that this definition does not address the fact that the malicious player may substitute his input for another value. We refer the reader to the original paper to see how this is addressed.

The fact that $\mathcal{R}$ is dependent on the particular adversary is a drawback to this definition, and unfortunately it is unavoidable in the gradual release approach. Intuitively, $\mathcal{R}$ must know whether $\mathcal{A}$ has recovered the output (using brute-force) in order to decide whether it must do the same. Otherwise, suppose $\mathcal{R}$ were fixed and the code of $\mathcal{A}$ were allowed to depend on $\mathcal{R}$; $\mathcal{A}$ could simply choose to abort the protocol and recover the output *just before* the point at which $\mathcal{R}$ would choose to do the same. $\mathcal{A}$ would still be polynomial time, and the fairness definition would be violated.

Although he does not provide the protocol, Yao also claims to have a solution that achieves this definition while enabling the computation of any polynomial-time function. The basic approach is to jointly compute an encryption of the output under public key $N = p \cdot q$ while revealing p to one player and q to the other. Then the players can run an exchange protocol for p and q.

Brickell, Chaum, Damgård, and van de Graaf, 1987: This Crypto 1987 paper [17] by Brickell et al. gives a new Blum-style approach to exchanging secret keys. They work in a large prime order group, $\mathbb{Z}_p^*$, under the (well accepted) assumption that it is hard to find discrete logarithms. They demonstrate a protocol for proving that the discrete logarithm of some group element lies in a particular interval. By repeatedly narrowing this interval, one can gradually release a secret. One obvious advantage over Blum's protocol [15], of course, is that their assumptions still stand up to cryptanalysis. Another advantage stressed by the authors is that this more easily enables broadcasting the release of the secret to a group, rather than releasing it to only one other party. This is because the receiver's part in the protocol is only to send random challenge bits (rather than squares for which only he knows the roots).

The authors do not define fairness. In fact, they barely discuss the application of exchange, focusing instead on the one-sided release of a secret. However, one could interleave the role of several senders to achieve secret exchange. In line with contemporary results on *zero-knowledge proofs* [48, 46], they do prove that the protocol reveals nothing other than the intended interval containing the logarithm.

Imagliazzo and Yung, 1987: The focus of this paper is not the topic of fairness, but zero knowledge. Impagliazzo and Yung [55] demonstrate a protocol for proving that you have correctly computed a circuit. However, they point out that their result can be used to prove correctness of the computation one bit at a time, even adaptively choosing which bit to reveal

next. If the output is a decryption key, this becomes the same approach to gradual release that Yao proposed [89]. It could also be used to reveal bits of the output directly (instead of to reveal a secret key that allows decryption of the encrypted output). Whether this is fair may depend heavily on the function being computed, since some output bits may be much more valuable than others.

It is interesting that the preceding breakthroughs in the area of zero-knowledge [48, 46], which lead to breakthroughs in secure computation [45, 37], also enable gradual release of verifiable secrets (in a very general form). Specifically, suppose the secret is verifiable via some string w, known only to the holder of the secret. Consider the statement: "the ith bit of the secret is b." This is an instance of an NP language for which w is the witness. Furthermore, as the authors of this paper show, if the secret is the result of computing some circuit (e.g., the circuit that multiplies large prime numbers), then the inputs and randomness used in the computation of that circuit suffice as a witness w.

Damgård, 1993: The main focus of this paper [28] is to enable efficient fair secret exchange for a wider variety of secrets. Prior schemes employing gradual release only enabled the exchange of a very particular type of secret: Blum [15], and Yao [89] suggest ways to exchange factors, and Brickell et al. [17] demonstrate how to exchange discrete logs.[8] In this work, Damgård introduces a new unconditionally hiding bit commitment, with length that is independent of the underlying value, along with a simple way of correctly revealing the committed value one bit at a time. Of course, for the protocol to be useful in some application, the players still need to prove that the committed value is relevant to the application, and not just some random string. Damgård demonstrates how to do this when the secret is a particular type of signature (such as RSA or El Gamal).

Damgård only defines security for a one-sided gradual release scheme. He requires a proof through simulation that nothing other than the last i bits of s is learned after i rounds. More formally, he includes the following requirement in his definition of security (rewritten in our own notation).

Definition 1.2.2 *Let π be a secure release protocol in which an adversary $\mathcal{A}$ gradually receives a secret s from player $\mathcal{H}$. Let $\mathcal{A}$ hold inputs $(1^\kappa, z, w)$, where z is some auxiliary input and w is a public value enabling the verification of s. Let $\mathcal{H}$ hold inputs $(1^\kappa, w)$. We denote by VIEW_π^i the view of $\mathcal{A}$ in this interaction after i rounds, and we denote by $s_{|i}$ the last i bits of s. π is fair if for every such adversary and for every round i there exists a simulator $\mathcal{S}(z, w, s_{|i})$ outputting $\text{VIEW}_\mathcal{S}^i$ such that $\text{VIEW}_\pi^i \stackrel{c}{\equiv} \text{VIEW}_\mathcal{S}^i$.*

Boneh and Naor, 2000: Boneh and Naor [16] give a result that is similar to Damgård's, building a commitment scheme that can be opened gradually. However, there are a few strong advantages to the newer scheme. For one, the commitments are designed to hold up against parallel algorithms: assuming some particular number theoretic assumption, the commitments are no easier to open with multiple processors than they are with a single processor. A second advantage is that a committed player can make it gradually easier to open the commitment without revealing any bits of the committed value. In contrast, in Damgård's commitment scheme, the committed player actually reveals a bit of the committed value in order to gradually open the commitment. When using such a scheme for releasing signatures, one would

[8] Merkle puzzles allow players to exchange a secret, but not in a provable fashion. Even et al. [33] did use Merkle puzzles to exchange secrets in a provable fashion, but only when the secrets were of a very specific form. The work of Impagliazzo and Yung [55] is actually very general, but it uses generic zero-knowledge proofs for NP, which involve a Karp reduction and are quite impractical.

have to assume that a single bit of the signature only makes it twice as easy to produce a valid signature. Here, this assumption is not necessary (though it is replaced with other number theoretic assumptions). In fact, the authors demonstrate that their protocol is zero knowledge, as long as the distinguisher runs in less time than is necessary to recover the committed value using brute-force.

As before, the commitment is only useful when we can prove something about the committed value. The authors demonstrate how to prove that the underlying committed value is an RSA signature. (They also demonstrate applications to other spheres of cryptography that are of lesser interest to our work.) They define fairness for the application of signature exchange in a way similar to that of Yao:

Definition 1.2.3 *A protocol is* (c, ϵ)-fair *if the following holds: for any adversary* $\mathcal{A}$ *running in time* $\mathsf{runtime}_{\mathcal{A}} < \kappa$*, let* $\mathcal{A}$ *choose a contract* C *and run the contract signing protocol with party* $\mathcal{H}$*. At some point,* $\mathcal{A}$ *aborts the protocol and attempts to recover a valid signature* $\sigma_{\mathcal{H}}(C)$*. Denote* $\mathcal{A}$*'s probability of success by* $\mathsf{success}_{\mathcal{A}}$*. Suppose now that party* $\mathcal{H}$ *runs the recovery algorithm* $\mathcal{R}$ *for time* $c \cdot \mathsf{runtime}_{\mathcal{A}}$ *and let* $\mathsf{success}_{\mathcal{H}}$ *be the probability he recovers a signature* $\sigma_{\mathcal{A}}(C)$*. Then* $\mathsf{success}_{\mathcal{A}} - \mathsf{success}_{\mathcal{H}} \leq \epsilon$*.*

The authors also give the first lower bound on the round complexity of a gradual release protocol. Letting $\gamma_{\mathcal{A}} = \frac{\mathsf{runtime}_{\mathcal{A}}}{\mathsf{success}_{\mathcal{A}}}$ and $\gamma_{\mathcal{H}} = \frac{\mathsf{runtime}_{\mathcal{H}}}{\mathsf{success}_{\mathcal{H}}}$, they define an *unfairness measure*: $\frac{\gamma_{\mathcal{A}}}{\gamma_{\mathcal{H}}}$. Then, given a signature scheme, they define its *security gap*, Δ, as the ratio between the time needed to forge a signature, and the time required to legitimately sign and verify. They prove that any κ-round gradual release protocol for exchanging signatures with security gap Δ has unfairness $\Delta^{1/\kappa}$.

Garay and Pomerance, 2003: Garay and Pomerance [40] improve on the scheme of Boneh and Naor [16] in several ways. First, when using the timed commitments of Boneh and Noar for signature exchange, the signatures and commitments have to have the same public modulus, which limits the types of signatures that can be exchanged. The protocol in this paper allows for the exchange of several types of digital signatures. Another benefit that they introduce, building on prior work by Garay and Jakobsson from the previous year [38], is to allow for the reuse of public keys for the commitments. This enables them to push the overhead required in Boneh and Naor's commitments to a single one-time setup. They can then recommit to new values using a very short, efficient protocol.

Pinkas, 2003: The three prior works on time-released commitments [28, 16, 40] all demonstrate ways of proving that the underlying committed value is of a particular form, such as, say, an RSA signature. Pinkas generalized the preceding works, providing a method for using timed commitments in general secure two-party computation [77]. Specifically, he takes Yao's approach for general secure computation [88], in which P_1 creates a "garbled circuit" for P_2 to evaluate. At the end of the protocol, each player has a commitment to the value of each output wire belonging to the other player.[9] They then gradually open their commitments. Pinkas uses a mix of "cut and choose" and *blind signatures* to enable the players to prove that the underlying committed values are the actual outputs from the computation. We refer the reader to the paper for details. His definition of fairness is very similar to the one in Boneh and Noar's work [16], so we do not present it here.

[9] Actually, this is a slightly inaccurate simplification. At the end of Yao's protocol, for every output wire i, P_2 learns a random key k_i representing the output bit on that wire, and only P_1 knows the mapping $k_i \to \{0, 1\}$. P_2 is supposed to send the random values corresponding to P_1's output wires, while P_1 simultaneously sends the mapping of $k_i \to \{0, 1\}$ for P_2's output wires. Pinkas demonstrates a way to compute Yao's circuits such that in the end, P_2 holds commitments to P_1's output bits, rather than just a random value associated with each output bit, and similarly that P_1 has a commitment to the mapping of $k_i \to \{0, 1\}$.

Garay, MacKenzie, Prabhakaran and Yang, 2006: This paper is the first work to place the gradual release approach into the formal security setting described in Chapter 2. The fairness of the protocol presented in the paper uses the same timed commitments presented in Garay and Pomerance's prior work in order to gradually release the output of any general secure computation [40]. The bigger contribution, though, is a new security framework, extending the framework of Chapter 2, in order to include a notion they call *resource fairness*.[10] Garay et al. compare the real world protocol to an ideal world protocol, as we will do in Chapter 2, but they modify the ideal world to model recovery of committed messages by the adversary through the use of brute-force. Specifically, they allow the ideal world adversary early access to these messages by "investing" resources, and then allow the honest party to request similar resources in order to recover fairness. They then prove that an execution by the simulator in this ideal model is indistinguishable from their real world protocol.

Probabilistic Fairness

There are several drawbacks to gradual release, which were very nicely described by Ben-Or et al. [12] (their own approach is described later):

1. It requires both parties to have the same resources.
2. There are no explicit instructions of what to do when a player aborts prematurely.
3. It relies on the "ideal" OWF assumption: given $f(x)$, even after revealing some bits of x, $f(x)$ is hard to invert.

In this section we present several works that took a second approach. As opposed to the technique of gradual release, where the solution becomes cryptographically easier to find after each round, they suggest using a probabilistic method in which the players' confidence in the solution increases over time. This has the advantage of addressing the first and third concerns described above, though as we will see it does not always handle the second concern.

Luby, Micali and Rackoff, 1983: Luby, Micali and Rackoff published a paper at FOCS in 1983 on the fair exchange of a bit [69].[11] Informally, each player creates a "coin" that is biased towards the value of their input bit by a factor of $1/\kappa$. They then take turns flipping each other's biased coin, each time gaining more confidence in the value of the other player's input. Actually, implementing it this way with two independent coins is insecure, because two independent random walks will likely diverge, creating a point in the protocol where one player will have a constant advantage in confidence, independent of κ. Instead, the authors present a way to create *correlated* biased coins, such that they either both land on the favored side, or neither does.

Vazirani and Vazirani, 1983: Concurrently, Vazirani and Vazirani demonstrated a way for Bob to send Alice a bit, while fairly receiving a "receipt" that proves that he did so. The approach is very similar to that of Luby et al.[12] The intuition is to have Alice and Bob create

[10] Actually, they extend the more general Universal Composability framework introduced by Canetti [21], which models security in an interactive environment involving other protocols. For simplicity we do not describe that framework in this thesis.

[11] Occasionally cited under the title "The MiRackoLus Exchange of a Secret Bit".

[12] By the previous discussion about random walks, it is not clear that this result can be made into a fair bit exchange protocol by interleaving two executions. On the other hand, the authors do state that the protocol of Luby et al. could be made into a one bit disclosure scheme, albeit with poorer efficiency.

two biased coins in each round. For the first coin, Alice chooses at random whether it should be biased towards the value of Bob's bit or towards the opposite value. (We stress that she does not know the value of Bob's bit, so she does not know whether the coin is biased towards 0 or 1. But she does know whether it is biased towards the value of Bob's bit.) The second coin is biased towards a value of Alice's choice, and has a further property that allows Alice to predict its outcome. Bob cannot distinguish the two coins from one another. He flips the two coins simultaneously, and reports both outcomes to Alice. She checks for cheating by watching the coin she can predict, and she learns something about Bob's bit from the other coin. After each flip, she signs a receipt for Bob. These coins are built on the quadratic residuosity assumption, and we leave the reader to find the details in the paper.

Beaver and Goldwasser, 1989 Just as Yao formalized the notion of fairness implied by gradual release, Beaver and Goldwasser [11] formalize the fairness provided by Luby et al. [69]. They demonstrate a protocol for n parties to compute any boolean function, regardless of the number of corruptions, achieving privacy, validity, and the notion of fairness defined below. To ease the exposition, we only define fairness for two parties, though the original paper defines it for n players. Let π be a protocol for computing $\mathcal{F} = \left(f_\kappa^1(x,y), f_\kappa^2(x,y)\right)$. Let $\text{VIEW}_\pi^{\mathcal{H},i}(x,y)$ denote the view of honest player $\mathcal{H}$ with input $(1^\kappa, x)$ after i rounds of interacting with $\mathcal{A}$ who holds input $(1^\kappa, y)$ in protocol π; we define $\text{VIEW}_\pi^{\mathcal{A},i}(x,y)$ similarly. Let $\mathsf{Correct}_{\mathcal{H}|i}(1^\kappa, x)$ denote the probability (taken over the random tapes, and the input distribution) that the honest player $\mathcal{H}$, holding input $(1^\kappa, x)$, outputs the correct output, $f_\kappa^1(x,y)$ after i rounds of interaction with $\mathcal{A}(1^\kappa, y)$ in protocol π. In other words, $\Pr[\mathsf{Correct}_{\mathcal{H}|i}] = \Pr[\text{OUT}_\pi^{\mathcal{H}} = f_\kappa^1(x,y) \mid \text{VIEW}_\pi^{\mathcal{H},i}(x,y)]$. We define $\mathsf{Correct}_{\mathcal{A}|i}$ similarly as the probability that the adversary, holding input $(1^\kappa, y)$ outputs the correct value $f_\kappa^2(x,y)$ after i rounds of interacting with $\mathcal{H}(x,y)$ in π. Below, to make the notation more succinct, we leave the inputs implicit.

Definition 1.2.4 *A protocol π for computing $\mathcal{F} = \left(f_\kappa^1(x,y), f_\kappa^2(x,y)\right)$ is considered d-*fair *if the following holds for all rounds i of π:*

$$\begin{aligned}
&\frac{1}{(1+\kappa^d)} \cdot \frac{\Pr\left[\mathsf{Correct}_{\mathcal{A}|0}\right]\left(1-\Pr\left[\mathsf{Correct}_{\mathcal{H}|0}\right]\right)}{\Pr\left[\mathsf{Correct}_{\mathcal{H}|0}\right]\left(1-\Pr\left[\mathsf{Correct}_{\mathcal{A}|0}\right]\right)} \\
&\qquad \leq \quad \frac{\Pr\left[\mathsf{Correct}_{\mathcal{A}|i}\right]\left(1-\Pr\left[\mathsf{Correct}_{\mathcal{H}|i}\right]\right)}{\Pr\left[\mathsf{Correct}_{\mathcal{H}|i}\right]\left(1-\Pr\left[\mathsf{Correct}_{\mathcal{A}|i}\right]\right)} \\
\leq \quad &(1+\kappa^d) \cdot \frac{\Pr\left[\mathsf{Correct}_{\mathcal{A}|0}\right]\left(1-\Pr\left[\mathsf{Correct}_{\mathcal{H}|0}\right]\right)}{\Pr\left[\mathsf{Correct}_{\mathcal{H}|0}\right]\left(1-\Pr\left[\mathsf{Correct}_{\mathcal{A}|0}\right]\right)}
\end{aligned}$$

A protocol is fair *if it is d-fair for all $d > 0$.*

Intuitively, the idea behind the definition is to require that the partial view of the players in the protocol never helps one player to guess the output much more than it helps the other (even though the adversarial player may be one round "ahead"). Because one player may begin the protocol with a greater advantage than the other (depending on the function and the input distribution), Beaver and Goldwasser compare the ratio of the probabilities of guessing the correct output before the protocol begins to the same ratio part-way through the protocol. (Technically, the above requirement only guarantees security when the first player is corrupt. We omit a similar definition for the case where the second player to act is corrupt.)

As mentioned, the authors present a protocol that achieves this definition for any polynomial-time computable boolean function. The intuition is similar to that used by Luby et al. [69] and

Vazirani et al. [87] for bit exchange. In each round, the players receive the output, XORed with a random bit that is biased slightly towards 0. Over many such rounds, their confidence in the output approaches 1. Technically, while the earlier works had to build such a scheme from scratch from the quadratic residuosity assumption, here the authors make use of the strong general results for secure computation that were developed only after 1983. Specifically, by 1989 it was known that the players can *unfairly* compute any function, including the function that outputs $\mathcal{F}(x, y) \oplus b$, where b is slightly biased towards 0.

With respect to the criticism of gradual release by Ben-or et al. [12], this approach, like [69, 87] before it, addresses two out of three problems. It removes the constraint that players have to have similar computational ability (concern 1). It also removes the strong assumptions about the impact of revealing part of a secret key (concern 3). On the surface, it seems to also address concern 2, that the decision of whether to recover the output is left external to the protocol. In fact this is only partially addressed by this approach. It is true that brute-force cannot help here, and that the output is always well defined; these properties seem beneficial. However, the output now comes with an associated measure of confidence, and the decision of whether to *trust* the output is external to the protocol.

Cleve, 1989: Cleve presents here a new approach to probabilistically revealing (or exchanging) a secret bit [26]. In the three protocols described above [69, 87, 11], at each round we could only describe the *expected* confidence of the receiving player. Here, Cleve presents a protocol such that at round i, the probability of guessing the sender's bit is always exactly p_i, where $p_1, \ldots, p_r$ are specified as parameters. The protocol is much more efficient in terms of round complexity, requiring r rounds to achieve the same level of confidence that the prior "biased-coin" protocols achieve in $r^2 \log^2(r)$ rounds (and even then, only in expectation). He also points out that through the now existing general techniques for secure computation, this scheme can be swapped in for Beaver and Goldwasser's fair release of the output of a computation, giving a more efficient protocol for any polynomial-time boolean function.

The definition of fairness he gives is quite informal, and only applies to the release of a single bit. He assumes that there is no prior knowledge, and that the input bit is uniformly distributed (though he says something informal about what happens when this is not true).

Goldwasser and Levin, 1990: The next year, Goldwasser and Levin [47] published a paper in which they refined the notion of fairness presented in [11], and introduced a new protocol. The protocol satisfies their definition of partially fair computation for any polynomial time function, whereas the four works above only allow for boolean output. Their notion of fairness is very similar to the definition by Beaver and Goldwasser. It is more technically involved, presumably addressing some issues that were left out of the previous definition. For example, it also requires a bound on the standard deviation of the ratio $\frac{\mathsf{Correct}_{\mathcal{H}|i}}{1-\mathsf{Correct}_{\mathcal{H}|i}}$ (as defined previously). Unfortunately, the authors do not spend time motivating the definition, or comparing it to the previous definition. It is hard to infer from the definition exactly what notion of fairness it achieves, and we do not present it here as the details do not greatly add to the bigger picture.

The bigger contribution of the work is to extend the approach to more general functions. Assume for the moment that the players receive the same output value OUT. The authors achieve fairness by having the players first compute $\text{OUT} \oplus \omega \bmod 2$, where ω is a random bit string of length $l = |\text{OUT}|$. Then, one round at a time, the players help each other to learn ω. They do this in l phases, each having κ steps. In the ith phase, the players slowly learn the dot product $\mathbf{v}_i \cdot \omega$, for random $\mathbf{v}_i \in \mathbb{Z}_2^{|\text{OUT}|}$, gaining confidence in the value over κ steps.

They repeat this for l such strings, $\mathbf{v}_1, \ldots, \mathbf{v}_l \in \mathbb{Z}_2^l$, revealing $\mathbf{v}_i \cdot \omega$ in phase i. The idea is to ensure that after every κ steps the search space for ω (information theoretically) is cut in half. Certainly this seems true if nothing is known about the output. However, notice that if an adversary has some *auxiliary input*, which reveals, say, some bits of the output, it is possible that a single dot product, $\mathbf{v}_i \cdot \omega$, cuts his search space by much more than half. This issue of auxiliary input is another concern that we will address when we define partial fairness in Chapter 5.

Rabin, 1981: The works above all use the same basic approach. They obscure the true answer with noise, and then reduce the noise over the course of the protocol, increasing the players' confidence in the output. In his earlier work on fair exchange, Rabin suggests a different, probabilistic approach to signature exchange [80]. He introduces a trusted third party, but tries to minimize it's involvement. Specifically, he suggests using a *beacon* that broadcasts signed, time-stamped random numbers from the interval $\{1, \ldots, \kappa\}$ at fixed time periods, but which has no other interaction with the players. The protocol works as follows. Alice and Bob begin by choosing a random value $1 \leq i \leq \kappa$. Alice signs and sends the following message: "I'm committed to the following contract as long as Bob can produce a signature from the beacon on time-stamp t and integer i." Bob responds with the symmetric message. If the beacon happens to sign i during time interval t, the protocol ends. Otherwise they choose a new random number and repeat the process. If Bob ever chooses to abort early, his advantage is exactly $1/\kappa$.

By changing the beacon slightly, Rabin also introduces a protocol for releasing private information in exchange for a receipt. (It seems this could be generalized to enable the fair exchange of two secrets, though Rabin does not address this.) The beacon broadcasts κ fresh encryption keys, $pk_1, \ldots, pk_\kappa$, at each interval t. In interval $t + \Delta$, he broadcasts a single corresponding decryption key, chosen at random. He then repeats these steps in a cycle. Alice and Bob again agree on a random value $1 \leq i \leq \kappa$. Alice encrypts the secret information using the ith public key that was broadcast at time t. As in the protocol for signature exchange, Bob's produces a signature on the statement: "I admit I have received the expected information, if and only if the beacon broadcasts the decryption key for pk_i at time-step $t + \Delta$." (Technically, we have to deal with a bad encryption by Alice, but Bob can stipulate this in his "admission". We omit these details.)

Ben-Or, Goldreich, Micali and Rivest: The ideas contained in this paper were first mentioned in a rump session talk by Rivest in 1981, though the paper was not published until 1985 [12]. As mentioned at the start of this section, one of the contributions of the work was to highlight the drawbacks of gradual release.[13] While the results of Luby et al. [69] and Vazirani et al. [87], which preceded this work, address some of the concerns, they only apply to bit exchange. The solution offered here is of a similar nature, and instead applies only to signature exchange. They redefine a signature such that it only verifies with some specified probability. Specifically, when signing a contract, the signer writes "The court should only accept this as a valid signature with probability p." The players alternate sending signatures of this form, increasing the value of p at each round until eventually p is overwhelmingly close to 1. If a player ever terminates early, the other brings the last contract he received to a judge. The judge flips a coin (having appropriate bias) and with probability p he rules the contract binding. With the remaining probability he rules that it is not binding.

[13] The authors did not appreciate the drawbacks of gradual release in 1981, which was why they did not bother to publish this paper until several years later.

Although this notion of a signature is unappealing, a major contribution of this paper is to point out the problems with prior approaches. More importantly, it is also the earliest work (chronologically) to formally define what is meant by fairness! In all earlier works, authors proved security of the protocols by showing that players would be caught if they deviated from the instructions. However, they do not prove anything about what the protocols actually achieve – this is left to the reader to evaluate. Ben-Or et al. begin by (informally) considering the following two definitions for fairness:

1. The probability that A is committed is large and the probability that B is not committed is small.

2. Conditioned on A being committed, the probability that B is not committed is small.

Certainly the second suggestion implies the first, so it is at least as strong. However, the authors point out that it is possible to satisfy the first without satisfying the second. They consider Rabin's protocol with beacons [80], where, using the beacon described previously for contract signing, after every message that Alice sends, and before Bob responds, the probability that Alice is committed and Bob is not is at most $1/\kappa$, satisfying definition 1. On the other hand, after Alice sends a message, and before Bob responds, *conditioned on Alice now being committed*, the probability that Bob is not committed is 1. The problem is that Bob is never committed before he responds! Under the first definition, this is okay because Alice is only committed with probability $1/\kappa$ anyway. But a protocol with this property cannot satisfy the second definition. They accordingly define the following notion of fairness. Let a player be *privileged* if they are capable of recovering a valid signature (perhaps with the help of a judge).

Definition 1.2.5 *A contract signing protocol is (v, ϵ)-fair for A if the following holds, for any contract $\mathcal{C}$, when A follows the protocol properly. At any step of the protocol in which the probability that B is privileged is greater than v, the conditional probability that A is not privileged, given that B is privileged, is at most ϵ. A protocol is (v, ϵ)-fair if it is (v, ϵ)-fair for both A and B.*

Then, using the approach described above, they simply instruct the players to use probability values for their signatures that are designed to meet this definition.

Fairness For an Honest Majority

At FOCS 1985, Chor et al. [23] introduced a primitive called verifiable secret sharing (VSS) which would have a huge impact on the field of secure computation. In standard t-out-of-n secret sharing schemes (as defined by Shamir [84]), a user can share a secret among n other players, with the guarantee that nothing is learned about the secret unless at least t of them cooperate to reconstruct it. VSS adds an additional security guarantee that enables the players (together with the owner of the secret) to verify that the shares provided are legitimate: i.e. that they reconstruct some unique, valid secret.

This was first used to guarantee fairness in general secure computation by Goldwasser, Micali and Wigderson [45], and then in a long line of following work [37, 22, 13, 81, 9]. The observation is that when $t > n/2$ players are honest, the players can safely begin the protocol by first verifiably sharing their inputs and their random tapes. Then, if a player later aborts the protocol, the honest players can cooperate to recover these values, and continue the protocol on his behalf. It follows from these results that complete fairness can be achieved as long as a strict majority of players are honest. (We are assuming here, and throughout this work,

that the players have access to a broadcast channel. This can be simulated using a public key infrastructure. In an information theoretic setting, and without a broadcast channel, the above results apply as long as more than $2/3$ of the players are honest.)

It is interesting that the goal of Chor et al. was to implement a simultaneous broadcast channel (which they also introduced). This is a communication network in which all players can speak simultaneously, with the guarantee that a) if a message is broadcast, then all players receive the same message, and b) if multiple messages are broadcast simultaneously, then the messages are independent. This second property does not hold in standard communication networks like the Internet, where it is quite easy to claim to have spoken simultaneously while actually delaying the sent message, reading the received messages, and then responding only afterwards. The authors make strong claims about what such a network would enable us to do:

> Simultaneous broadcast networks are a fundamental primitive as simultaneity lies at the heart of many protocols: coin flipping, fair voting, contract signing, exchanging secrets etc... All these protocols are in fact extremely easy to implement in an simultaneous broadcast network.

In fact, this is not true – the only application among the list that is easy to implement using a simultaneous broadcast channel is coin flipping. (To implement this, players can all broadcast a random bit simultaneously, and output the XOR of all received bits, using default values for any players that abort.) The difficulty with performing, say, two-party signature exchange in the same manner is that you have no way of ensuring that the other player will send the correct value. We prove in Chapter 7 that simultaneous broadcast is not complete for fairness (i.e., it does not allow us to compute all functions fairly). Indeed, we demonstrate this by proving that a particular function, which is similar to signature exchange, cannot be fairly computed even when players have access to a simultaneous broadcast channel. It is an excellent open question to figure out exactly which functions this primitive allows us to fairly compute. Informally, our own conjecture is that it does not help for very many functions beyond coin flipping.

Optimistic Exchange

A different notion of fairness was suggested by Asokan et al. [3] and Micali [72]. Here, the users interact in a protocol that may end unfairly, but, whenever it does, they are guaranteed that fairness can be restored by a trusted third party. The novelty here is that the trusted party can remain offline until their is a dispute. This notion of fairness is very interesting, and has lead to a long line of research, both on optimistic contract signing in the two-party case [4, 8, 39, 76, 5], the multi-party case [30, 66, 65], and on optimistic general secure computation [19]. We do not review these works here.

Chapter 2

Definitions and Preliminaries

In this chapter we provide some basic definitions and notations that are used throughout the thesis. In Sections 2.1 through 2.7 we define standard notation and the conventional notions of security in secure computation [60, 42]. The reader familiar with this material may choose to skip those subsections, or return to them as needed. In Section 2.8 we present a canonical form for secure computation that almost all of the protocols in this thesis will follow. We note that some additional definitions that are used only in Chapters 6 and 7 appear in those chapters rather than here.

2.1 Basic Notation

We let κ denote the security parameter. A function $\mu(\cdot)$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < 1/p(\kappa)$. A *distribution ensemble* $X = \{X(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_\kappa$ and $\kappa \in \mathbb{N}$, where $\mathcal{D}_\kappa$ is a set that may depend on κ. (Looking ahead, κ will be the security parameter and $\mathcal{D}_\kappa$ will denote the domain of the parties' inputs.) Two distribution ensembles $X = \{X(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ are *computationally indistinguishable*, denoted $X \stackrel{\text{c}}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every κ and every $a \in \mathcal{D}_\kappa$

$$\left|\Pr[D(X(a,\kappa)) = 1] - \Pr[D(Y(a,\kappa)) = 1]\right| \leq \mu(\kappa).$$

The statistical difference between two distributions $X(a,\kappa)$ and $Y(a,\kappa)$ is defined as

$$\mathsf{SD}\left(X(a,\kappa),\, Y(a,\kappa)\right) = \frac{1}{2}\cdot\sum_s \left|\Pr[X(a,\kappa) = s] - \Pr[Y(a,\kappa) = s]\right|,$$

where the sum ranges over s in the support of either $X(a,\kappa)$ or $Y(a,\kappa)$. Two distribution ensembles $X = \{X(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ are *statistically close*, denoted $X \stackrel{\text{s}}{\equiv} Y$, if there is a negligible function $\mu(\cdot)$ such that for every κ and every $a \in \mathcal{D}_\kappa$, it holds that $\mathsf{SD}\left(X(a,\kappa), Y(a,\kappa)\right) \leq \mu(\kappa)$.

Functionalities. In the two-party setting, a *functionality* $\mathcal{F} = \{f_\kappa\}_{\kappa\in\mathbb{N}}$ is a sequence of randomized processes, where each f_κ maps pairs of inputs to pairs of outputs (one for each party). We write $f_\kappa = (f_\kappa^1, f_\kappa^2)$ if we wish to emphasize the two outputs of f_κ, but stress that if f_κ^1 and f_κ^2 are randomized then the outputs of f_κ^1 and f_κ^2 are correlated random variables. The domain

of f_κ is $X_\kappa \times Y_\kappa$, where X_κ (resp., Y_κ) denotes the possible inputs of the first (resp., second) party. If $|X_\kappa|$ and $|Y_\kappa|$ are polynomial in κ, then we say that $\mathcal{F}$ is defined over *polynomial-size domains*.

The above definition extends naturally to the multi-party setting. In this thesis, the only multi-party functions we will consider have the form $f : \{0,1\} \times \cdots \times \{0,1\} \to \{0,1\}$, where the input of player P_i is x_i, and all n players receive the same output bit.

2.2 Basic Cryptographic Primitives

Message authentication codes. We briefly review the standard definition for information-theoretically secure message authentication codes (MACs). A *message authentication code* consists of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$. The *key-generation algorithm* Gen takes as input the security parameter 1^κ in unary and outputs a key k. The *message authentication algorithm* Mac takes as input a key k and a message $M \in \{0,1\}^{\leq n}$, and outputs a tag t; we write this as $t = \mathsf{Mac}_k(M)$. The *verification algorithm* Vrfy takes as input a key k, a message $M \in \{0,1\}^{\leq n}$, and a tag t, and outputs a bit b; we write this as $b = \mathsf{Vrfy}_k(M,t)$. We regard $b = 1$ as acceptance and $b = 0$ as rejection, and require that for all κ, all k output by $\mathsf{Gen}(1^\kappa)$, and all $M \in \{0,1\}^{\leq \kappa}$, it holds that $\mathsf{Vrfy}_k(M, \mathsf{Mac}_k(M)) = 1$.

We say $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ is a *secure m-time MAC*, where m may be a function of κ, if no computationally unbounded adversary can output a valid tag on a new message, with high probability, after seeing valid tags on m other messages. For our purposes, we do not require security against an adversary who adaptively chooses its m messages for which to obtain a valid tag; it suffices to consider a non-adaptive definition where the m messages are fixed in advance. (Nevertheless, known constructions satisfy the stronger requirement.) Formally:

Definition 2.2.1 *Message authentication code* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an* information-theoretically secure m-time MAC *if for any sequence of messages $M_1, \ldots, M_m$ and any adversary $\mathcal{A}$, the following is negligible in the security parameter κ:*

$$\Pr\left[\begin{array}{l} k \leftarrow \mathsf{Gen}(1^\kappa);\ \forall i : t_i = \mathsf{Mac}_k(M_i); \\ (M', t') \leftarrow \mathcal{A}(M_1, t_1, \ldots, M_m, t_m) \end{array} \ :\ \mathsf{Vrfy}_k(M', t') = 1 \bigwedge M' \notin \{M_1, \ldots, M_m\}\right].$$

Digital signatures. In some cases where we consider multi-party computation it will not suffice for us to rely on message authentication codes. (With MACs, the ability to verify implies the ability to tag, and this causes problems when there is a possibility of collusion.) We therefore review the notion of a *digital signature*, which is the public-key counter-part of MACs. Digital signatures are known to exist assuming one way functions exist [83].

A digital signature scheme consists of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. The *key-generation algorithm*, Gen, takes as input the security parameter 1^κ in unary and outputs a pair of keys (pk, sk). The *signing algorithm*, Sign, takes as input a key sk and a message $M \in \{0,1\}^{\leq n}$, and outputs a signature σ; we write this as $\sigma = \mathsf{Sign}_{sk}(M)$. The *verification algorithm*, Vrfy, takes as input a key pk, a message $M \in \{0,1\}^{\leq n}$, and a signature σ, and outputs a bit b; we write this as $b = \mathsf{Vrfy}_{pk}(M, \sigma)$. We regard $b = 1$ as acceptance and $b = 0$ as rejection, and require that for all κ, all (pk, sk) output by $\mathsf{Gen}(1^\kappa)$, all $M \in \{0,1\}^{\leq n}$, it holds that $\mathsf{Vrfy}_{pk}(M, \mathsf{Sign}_{sk}(M)) = 1$.

As with the case of MACs, it suffices for us to have signature schemes that are m-time secure against a non-adaptive adversary.

Definition 2.2.2 *Signature scheme* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an* m-time signature scheme *if for any sequence of messages* $M_1, \ldots, M_m$ *and any probabilistic, polynomial-time adversary* $\mathcal{A}$*, the following is negligible in the security parameter* κ*:*

$$\Pr\left[\begin{array}{c}(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa);\ \forall i : \sigma_i = \mathsf{Sign}_{sk}(M_i); \\ (M', \sigma') \leftarrow \mathcal{A}(M_1, \sigma_1, \ldots, M_m, \sigma_m)\end{array} \ : \ \mathsf{Vrfy}_{pk}(M', \sigma') = 1 \bigwedge M' \notin \{M_1, \ldots, M_m\}\right].$$

Secret sharing schemes. A secret sharing scheme allows a dealer to split some secret $s \in \mathbb{F}$, where $\mathbb{F}$ is some publicly known field, into *shares*, such that reconstruction of s is possible only if enough shares are known.

Definition 2.2.3 *A* t*-out-of-*n *secret sharing scheme is a pair of polynomial-time algorithms,* $(\mathsf{Share}, \mathsf{Rec})$*. On input* $t, n \in \mathbb{N}$ *and* $s \in \mathbb{F}$*,* $\mathsf{Share}(t, n, s)$ *outputs* shares $\{s_1, \ldots, s_n\} \in \mathbb{F}$ *with the following properties:*

- *For any* $S \subset \{s_1, \ldots, s_n\}$ *such that* $|S| < t$*,* S *reveals nothing about* s *information theoretically.*
- *For any* $S \subset \{s_1, \ldots, s_n\}$ *such that* $|S| \geq t$*,* $\mathsf{Rec}(S) = s$*.*

Non-malleable secret sharing schemes. A non-malleable secret sharing scheme is a secret sharing scheme with an additional property, guaranteeing that if any party manipulates their share in any way, the reconstruction protocol outputs a special failure symbol. For our purposes, a 2-out-of-2 scheme will suffice, so we define that.

Definition 2.2.4 *A* 2*-out-of-*2 *non-malleable secret sharing scheme (NMSS scheme) is defined by a pair of polynomial-time algorithms* $(\mathsf{Share}, \mathsf{Rec})$ *with the following properties:*

- $\mathsf{Share}(s, r)$ *returns* 2 *shares,* (s_0, s_1) *(where* s_i *is the share of the* i*-th party) such that a single share reveals no information about* s*.*
- $\mathsf{Rec}(\mathsf{Share}(s, r)) = (s, 0)$ *for every* s, r*. The second output of* Rec *serves as a flag which is set to 0 if the secret has been successfully reconstructed.*
- *Any attempt by a player to modify their share (independently of the remaining share) is detected with overwhelming probability. Formally, we say that* $(\mathsf{Share}, \mathsf{Rec})$ *is* ϵ*-non-malleable if for every secret* s*, every (computationally unbounded) adversary* $\mathcal{A}$ *can win the following game with probability at most* ϵ*:*
  - $\mathcal{A}$ *corrupts one of the parties.*
  - *Random shares* (s_0, s_1) *from* $\mathsf{Share}(s, r)$ *are given to the* 2 *parties.*
  - *Based on the share* $s_{\mathcal{A}}$ *it observed,* $\mathcal{A}$ *computes a new share* $s^*_{\mathcal{A}}$*.*
  - $\mathcal{A}$ *wins if* $s^*_{\mathcal{A}} \neq s_{\mathcal{A}}$ *and* $\mathsf{Rec}(s^*_{\mathcal{A}}, s_H) = (s', 0)$ *for some secret* s'*, where* s_H *is the share received by the uncorrupted party.*

2.3 Secure Two-Party Computation with Complete Fairness

In what follows, we define what we mean by a *secure* protocol. This definition follows the definition of [42], which is today the "accepted" notion of security[1]. The basic underlying idea of the definition is to compare the result of the interaction to an *ideal* world that is secure *by definition*. Specifically, we prove a protocol is secure by comparing it to an ideal world where the players submit their inputs to a trusted party that computes the functionality on their behalf, and simultaneously returns appropriate output to each participant. Of course, on the surface, our real world protocol and this ideal world are very different, so it is not immediately obvious what it means to compare them. Let us begin by considering our *privacy* requirement: the players should not learn anything more from the interaction than is revealed from their own inputs and outputs. The earliest security definitions formalized this requirement in the following way: for any probabilistic polynomial-time adversary $\mathcal{A}$ interacting in a protocol π for computing functionality $\mathcal{F}$, there must exist a probabilistic polynomial-time simulator $\mathcal{S}$ acting in the ideal world, where there is trusted access to functionality $\mathcal{F}$, such that

$$\left\{\text{VIEW}_{\pi,\mathcal{A}(x,\kappa)}(x,y,\kappa)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{VIEW}_{\mathcal{F},\mathcal{S}(x,\kappa)}(x,y,\kappa)\right\}$$

Here, $\text{VIEW}_{\pi,\mathcal{A}(x,\kappa)}(x,y,\kappa)$ is a random variable representing the view of the adversary when acting in the real world protocol π, where both he and the honest player are given security parameter κ, $\mathcal{A}$ is given input $x \in X_\kappa$, and the honest player is given input $y \in Y_\kappa$. The random variable $\text{VIEW}_{\mathcal{F},\mathcal{S}(x,\kappa)}(x,y,\kappa)$ is some transcript created by the simulator in the ideal world, after being given the same input as the adversary, along with the adversary's output from $\mathcal{F}(x,y)$. We know by definition that the simulator cannot have learned anything inappropriate in the ideal world. We can conclude that if the simulator can create the same view that $\mathcal{A}$ would see while interacting in protocol π, then the view of $\mathcal{A}$ is harmless as well.

This definition works very well for the privacy requirement, but what about our other security concerns? The above security notion ensures that the view of the adversary is harmless; our only other concern can be with the output of the honest player. Our worry here is that the adversary could somehow influence the view of the honest player in a way that results in "bad" output. For example, if the functionality is randomized, the adversary should not be able to skew the output distribution. So how exactly should we define "bad" output? One thing to note is that there is no way we can prevent the adversary from changing his own input: even if he is given input x, he can always choose to interact honestly as though he were given input x'. However, putting this issue aside, we can demand strong security properties by again comparing to an ideal world. Letting $\text{OUT}_{\pi,\mathcal{A}(x,\kappa)}(x,y,\kappa)$ be a random variable representing the output of the honest player on inputs y and κ while interacting with $\mathcal{A}(x,\kappa)$ in π, and $\text{OUT}_{\mathcal{F},\mathcal{S}(x,\kappa)}(x,y,\kappa)$ be a random variable representing the honest player's output in the ideal world, we have the following requirement:

$$\left\{\text{OUT}_{\pi,\mathcal{A}(x,\kappa)}(x,y,\kappa)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{OUT}_{\mathcal{F},\mathcal{S}(x,\kappa)}(x,y,\kappa)\right\}$$

This guarantees that regardless of how $\mathcal{A}$ behaves in the protocol π, the output distribution of the honest player is indistinguishable from his output distribution in the ideal world. However, we do not restrict the simulator $\mathcal{S}$ in terms of *what* values it may submit to the function-

[1] Actually, this definition of security only applies in the stand-alone setting, where the function being computed is isolated from other computations. The work of Canetti [20] demonstrates an even stronger notion of security that allows us to prove protocols secure even when they are executed simultaneously with arbitrary other protocols. For the purposes of our work, we are satisfied with the stand-alone setting.

ality $\mathcal{F}$ in the ideal world. In particular, it may choose to submit any $x' \in X_\kappa$, and, therefore, so may the adversary $\mathcal{A}$ in the real world.

We note that it does not suffice to consider these two security properties independently, but rather we are concerned with the joint distribution over the above random variables. This captures the requirement that the protocol not only protect the privacy of the inputs, but that it also guarantee that the adversary cannot correlate his choice of input with the input of the other party. This will be made explicit in the formal definition of security, which appears next. In this definition, we give the adversary some auxiliary information z. This models the possibility that each player will have some external information about the other player's input. It also allows us to model non-uniform computation. Technically, we should also prove that the protocol is secure against eavesdroppers. However, in the two-party case this property is easily achieved by using a secure channel (which can be assumed, or can be implemented using cryptography.) Finally, we note that the definition implies a very strong notion of fairness. This is inherited from the property of the ideal functionality that gives output to both players at exactly the same time. We are now ready for the complete definition.

Execution in the ideal model. The parties are P_1 and P_2, and there is an adversary $\mathcal{A}$ who has corrupted one of them. An ideal execution for the computation of $\mathcal{F} = \{(f_\kappa^1, f_\kappa^2)\}$ proceeds as follows:

Inputs: P_1 and P_2 hold the same value 1^κ, and their inputs $x \in X_\kappa$ and $y \in Y_\kappa$, respectively; the adversary $\mathcal{S}$ receives an auxiliary input z.

Send inputs to trusted party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{S}$ may send any value of its choice. Denote the pair of inputs sent to the trusted party by (x', y').

Trusted party sends outputs: If $x' \notin X_\kappa$ the trusted party sets x' to some default input in X_κ; likewise if $y' \notin Y_\kappa$ the trusted party sets y' equal to some default input in Y_κ. Then the trusted party chooses r uniformly at random and sends $f_\kappa^1(x', y'; r)$ to party P_1 and $f_\kappa^2(x', y'; r)$ to party P_2.

Outputs: The honest party outputs whatever it was sent by the trusted party. $\mathcal{S}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x, y, \kappa) = \big(\text{VIEW}_{\mathcal{F},\mathcal{S}(x,\kappa,z)}(x, y, \kappa), \text{OUT}_{\mathcal{F},\mathcal{S}(x,\kappa,z)}(x, y, \kappa)\big)$$

be the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Execution in the real model. We next consider the real model in which a two-party protocol π is executed by P_1 and P_2 (and there is no trusted party). In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted party and sends all messages on behalf of this party, using an arbitrary polynomial-time strategy. The honest party follows the instructions of π.

Let $\mathcal{F}$ be as above and let π be a two-party protocol computing $\mathcal{F}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. We let

$$\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, \kappa) = \big(\text{VIEW}_{\pi,\mathcal{A}(x,\kappa,z)}(x, y, \kappa), \text{OUT}_{\pi,\mathcal{A}(x,\kappa,z)}(x, y, \kappa)\big)$$

be the random variable consisting of the view of the adversary and the output of the honest party, following an execution of π where P_1 begins by holding 1^κ and input x and P_2 begins by holding 1^κ and y.

Security as emulation of an ideal execution in the real model. Having defined the ideal and real models, we can now define security of a protocol. Loosely speaking, the definition asserts that a secure protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated as follows:

Definition 2.3.1 *Protocol π is said to* securely compute $\mathcal{F}$ with complete fairness *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_\kappa\times Y_\kappa,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_\kappa\times Y_\kappa,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}}$$

Relationship to Fairness: The above definition requires that the fairness achieved in the real world protocol is indistinguishable from the complete fairness achieved (by definition) in the ideal world. This is much stronger than any of the definitions that are given in Section 1.2, and indeed, it turns out to be impossible to achieve in general when at least half of the players are malicious. However, as we shall see, it can be achieved for some interesting functions.

One interesting observation which we made in Section 1.2 is that this definition does still allow for one player to have an advantage with respect to *confidence* in the value of $\mathcal{F}(x,y)$. At first glance, this might not be obvious, since the definition requires that a secure protocol is indistinguishable from an ideal execution where both players simultaneously receive output, and learn nothing else. Consider, however, an ideal world adversary $\mathcal{A}$ that has input x but submits x' to the trusted party. It is quite possible that from the result $\mathcal{F}(x',y)$, $\mathcal{A}$ gains confidence in the "correct" output $\mathcal{F}(x,y)$. At the same time, the honest player outputs $\mathcal{F}(x',y)$ which may in fact be incorrect. Since this behavior is impossible to prevent, we see this as evidence that confidence in the output is not an appropriate measure for fairness.

2.4 Secure Two-Party Computation With Abort

As discussed in the previous chapter, we already know that certain functions of interest, such as bit exchange and signature exchange, cannot be achieved in the two-party setting according to Definition 2.3.1. As such, the following relaxation of the previous definition has become the conventional definition in the two-party setting [42]. It differs only in the definition of the ideal world, where it allows the malicious player to *abort early*. Specifically, the adversary will receive its own output from the functionality first, and then choose whether or not the functionality should give output to the honest party as well. By relaxing the ideal world model in this way, we are making it easier to prove security of real world protocols when we have aborting adversaries. We note that with the exception of fairness, all *other* security properties are still preserved by this definition. The benefit of this definition is that there are general results showing how to compute *any* polynomial-time computable function securely according to this definition [42].

We again let P_1 and P_2 denote the two parties, and consider an adversary $\mathcal{A}$ who has corrupted one of them. The only change from the definition in Section 2.3 is with regard to the ideal model for computing $\mathcal{F} = \{f_\kappa\}$, which is now defined as follows:

Inputs: As previously.

Send inputs to trusted party: As previously.

Trusted party sends output to corrupted party: If $x' \notin X_\kappa$ the trusted party sets x' to some default input in X_κ; likewise if $y' \notin Y_\kappa$ the trusted party sets y' equal to some default input in Y_κ. Then the trusted party chooses r uniformly at random, computes $z_1 = f_\kappa^1(x', y'; r)$ and $z_2 = f_\kappa^2(x', y'; r)$, and sends z_i to the corrupted party P_i (i.e., to the adversary $\mathcal{S}$).

Adversary decides whether to abort: After receiving its output (as described above), the adversary either sends abort or continue to the trusted party. In the former case the trusted party sends $\perp$ to the honest party P_j, and in the latter case the trusted party sends z_j to P_j.

Outputs: As previously.

We let $\text{IDEAL}^{\mathsf{abort}}_{\mathcal{F},\mathcal{A}(z)}(x, y, \kappa)$ be the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Definition 2.4.1 *Protocol π is said to* securely compute $\mathcal{F}$ with abort *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}^{\mathsf{abort}}_{\mathcal{F},\mathcal{S}(z)}(x, y, \kappa)\right\}_{(x,y)\in X_\kappa \times Y_\kappa,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, \kappa)\right\}_{(x,y)\in X_\kappa \times Y_\kappa,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}}$$

2.5 Secure Multi-Party Computation with Complete Fairness

Execution in the ideal model. The parties are $\mathcal{P} = \{P_1, \ldots, P_n\}$, and there is an adversary $\mathcal{S}$ who has corrupted some subset $\mathcal{I} \subset \mathcal{P}$ of them. An ideal execution for the computation of $\mathcal{F}$ proceeds as follows:

Inputs: Each party P_i holds its input x_i and the security parameter κ. The adversary $\mathcal{S}$ also receives an auxiliary input z.

Send inputs to trusted party: The honest parties send their inputs to the trusted party. $\mathcal{S}$ may substitute any values it likes on behalf of the corrupted parties. We denote by x_i' the value sent to the trusted party on behalf of P_i.

Trusted party sends outputs: If any x_i' is not in the correct domain, the trusted party sets $x_i' = \hat{x}_i$ for some default value $\hat{x}_i$. Then, the trusted party chooses r uniformly at random and sends $f_i(x_1', \ldots, x_n'; r)$ to each P_i.

Outputs: The honest parties output whatever they were sent by the trusted party, the corrupted parties output nothing, and $\mathcal{S}$ outputs an arbitrary function of its view.

We let $\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x_1, \ldots, x_n, \kappa)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model as described above.

Execution in the real model. Here a multi-party protocol π is executed by $\mathcal{P}$, and there is no trusted party. In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted parties (as well as an auxiliary input z) and sends all messages on behalf of these parties, using an arbitrary polynomial-time strategy. The honest parties follow the instructions of π.

Let $\mathcal{F}$ be as above and let π be a multi-party protocol computing $\mathcal{F}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. We let $\text{REAL}_{\pi,\mathcal{A}(z)}(x_1,\ldots,x_n,\kappa)$ be the random variable consisting of the view of the adversary and the output of the honest parties, following an execution of π where P_i begins holding its input x_i and the security parameter κ.

Security as emulation of an ideal execution. Having defined the ideal and real models, we can now define security of a protocol. Loosely speaking, the definition says that a secure protocol (in the real model) emulates the ideal model (in which a trusted party exists).

Definition 2.5.1 *Let $\mathcal{F}$ be as above. Protocol π is said to t-*securely compute *$\mathcal{F}$ with* complete fairness *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model that corrupts at most t players, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(\vec{x},\kappa)\right\}_{\vec{x}\in(\{0,1\}^*)^n,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(\vec{x},\kappa)\right\}_{\vec{x}\in(\{0,1\}^*)^n,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}}$$

Technically, we should also consider security against an external eavesdropper, even in the case when all players are considered honest. We assume for simplicity that there is a secure broadcast channel between the players, which immediately guarantees this security property[2].

2.6 Secure Multi-Party Computation With Designated Abort

This definition is standard for secure multi-party computation without an honest majority [42], and is a direct parallel to Definition 2.4.1. It allows *early abort* (i.e., the adversary may receive its own outputs even though the honest parties do not), but only if P_1 is corrupted.

We again let $\mathcal{P} = \{P_1,\ldots P_n\}$ denote the parties, and consider an adversary $\mathcal{S}$ who has corrupted a subset $\mathcal{I} \subset \mathcal{P}$ of them. The only change from the definition in Section 2.5 is with regard to the ideal model for computing $\mathcal{F}$, which is now defined as follows:

Inputs: As previously.

Send inputs to trusted party: As previously.

> If any x_i' is not in the correct domain, the trusted party sets $x_i' = \hat{x}_i$ for some default value $\hat{x}_i$. Then, the trusted party chooses r uniformly at random and sets $z_i = f_i(x_1',\ldots,x_n';r)$.

Trusted party sends outputs, P_1 honest: The trusted party sends z_i to each P_i.

Trusted party sends outputs, P_1 corrupt: The trusted party sends $\{z_i\}_{i:P_i\in\mathcal{I}}$ to $\mathcal{S}$. Then $\mathcal{S}$ sends either abort or continue to the trusted party. In the former case the trusted party sends $\perp$ to all honest parties, and in the latter case the trusted party sends z_i to each honest P_i.

> Note that an adversary corrupting P_1 can always abort the protocol, even if $|\mathcal{I}| < n/2$.

[2] To actually implement such a channel, players could (for example) encrypt each message $n-1$ times, once under each of the other player's public keys, and broadcast all $n-1$ messages. They then follow this with a zero-knowledge proof that each ciphertext is an encryption of the same message. In fact, it suffices to do this only once in order to establish a single shared symmetric key that can be used to encrypt all broadcast messages in the remainder of the protocol.

Outputs: As previously.

We let $\text{IDEAL}^{\text{abort}}_{\mathcal{F},\mathcal{S}(z)}(\vec{x},\kappa)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model as described above.

Definition 2.6.1 *Let f be a functionality, and let π be a protocol computing $\mathcal{F}$. Protocol π is said to* t-securely compute $\mathcal{F}$ with designated abort *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model that corrupts at most t players, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}^{\text{abort}}_{\mathcal{F},\mathcal{S}(z)}(\vec{x},\kappa)\right\}_{\vec{x}\in(\{0,1\}^*)^n,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(\vec{x},\kappa)\right\}_{\vec{x}\in(\{0,1\}^*)^n,\, z\in\{0,1\}^*,\, \kappa\in\mathbb{N}}$$

2.7 The Hybrid Model

The hybrid model combines both the real and ideal models. Specifically, an execution of a protocol π in the $\mathcal{G}$-hybrid model, for some functionality $\mathcal{G}$, involves the parties sending normal messages to each other (as in the real model) but, in addition, the parties have access to a trusted party computing $\mathcal{G}$. The parties communicate with this trusted party in exactly the same way as in the ideal models described above; the question of which ideal model is taken (that with or without abort) must be specified. In this manuscript, we will always considered a hybrid model where the functionality $\mathcal{G}$ is computed according to the ideal model *with abort*.

For our purposes here, we will require that any protocol in the $\mathcal{G}$-hybrid model makes only *sequential* calls to $\mathcal{G}$; i.e., there is at most a single call to $\mathcal{G}$ per round, and no other messages are sent during any round in which $\mathcal{G}$ is called.

Let $\mathcal{G}$ be a functionality and let π be a two-party protocol for computing some functionality $\mathcal{F}$, where π includes real messages between the parties as well as calls to $\mathcal{G}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. We let $\text{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A}(z)}(x,y,\kappa)$ be the random variable consisting of the view of the adversary and the output of the honest party, following an execution of π (with ideal calls to $\mathcal{G}$) where P_1 begins by holding 1^κ and input x and P_2 begins by holding 1^κ and input y. Both security with complete fairness and security with abort can be defined via the natural modifications of Definitions 2.3.1 and 2.4.1.

The hybrid model gives a powerful tool for proving the security of protocols. Specifically, we may design a real-world protocol for securely computing some functionality $\mathcal{F}$ by first constructing a protocol for computing $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Letting π denote the protocol thus constructed (in the $\mathcal{G}$-hybrid model), we denote by π^ρ the real-world protocol in which calls to $\mathcal{G}$ are replaced by sequential execution of a real-world protocol ρ that computes $\mathcal{G}$. ("Sequential" here implies that only one execution of ρ is carried out at any time, and no other π-protocol messages are sent during execution of ρ.) The results of [20] then imply that if π securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, and ρ securely computes $\mathcal{G}$ with abort, we can conclude that the composed protocol π^ρ securely computes $\mathcal{F}$ (in the real world). For completeness, we state this formally in the form we will use in this work:

Proposition 2.7.1 *Let ρ be a protocol that securely computes $\mathcal{G}$ with abort, and let π be a protocol that securely computes $\mathcal{F}$ with complete fairness in the $\mathcal{G}$-hybrid model (where $\mathcal{G}$ is computed according to the ideal world with abort). Then protocol π^ρ securely computes $\mathcal{F}$ with complete fairness.*

In Section 2.8 we will describe a functionality that we call ShareGen. This functionality will be very useful in the results presented in later chapters, as almost all of our protocols will begin by having the players perform some known, general protocol for the secure computation of ShareGen. In our proofs of security, we will make use of the preceding proposition, and rather than proving that the execution of ShareGen was secure, we will compare the ideal world to the hybrid world where the parties have ideal access to ShareGen.

2.8 A Canonical Form for Fair Two-Party Computation

We present here a general format that will be used in many of the results presented in this thesis. It is general enough that *any* protocol for two-party computation can be compiled to fit this format, and, in addition to simplifying the description of our protocols, it also makes it far easier to analyze their fairness properties. The protocol format first formally appeared in our work on complete fairness [49] which is described in Chapter 3, however a very similar idea was used even in our earlier work on rational secret sharing [51], presented in Chapter 6.

The basic format is as follows. To compute $\mathcal{F}(x, y)$, the players begin by using their inputs to $\mathcal{F}$ to compute a different functionality we call ShareGen. This will be done *unfairly* (i.e., using a protocol that achieves security with abort, as defined in Section 2.4) using known results for secure two-party computation [88, 45, 42]. Instead of returning $\mathcal{F}(x, y)$, $\mathsf{ShareGen}(x, y)$ returns to each player two sequences of *secret shares*, where the secrets are related to $\mathcal{F}(x, y)$ in some way. Exactly how the secrets are related to $\mathcal{F}(x, y)$ will depend on the particular function $\mathcal{F}$, and will have to be specified each time we present a new protocol. However, we will only need to redefine a small (isolated) component, and the remainder of the protocol will remain according to the template presented here. For now, it suffices to think of the output of ShareGen as secret shares of sequences of *plausible* outputs from $\mathcal{F}$. The description of ShareGen appears in Figure 2.1. We note that although the computation of ShareGen will be unfair, the output of ShareGen does not reveal anything about the output $\mathcal{F}(x, y)$ (due to the randomness of the secret sharings).

ShareGen

Inputs: Let the inputs to ShareGen be x and y. If one of the received inputs is not in the correct domain, it is replaced with a default value. The security parameter is κ.

Computation:

1. Choose the values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$.
   (Exact instructions for how to do this depend on the function $\mathcal{F}$ being computed.)
2. For $1 \leq i \leq m$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (I.e., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)
3. Compute $k_a, k_b \leftarrow \mathsf{Gen}(1^\kappa)$. For $1 \leq i \leq m$, let $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ and $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$.

Output:

1. P_1 receives the values $a_1^{(1)}, \ldots, a_m^{(1)}$ and $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and the MAC-key k_a.
2. P_2 receives the values $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$ and $b_1^{(2)}, \ldots, b_m^{(2)}$, and the MAC-key k_b.

Figure 2.1: Functionality ShareGen with parameter $m = m(\kappa)$.

$\Pi_{\mathcal{F}}$

Inputs: Party P_1 has input x and party P_2 has input y. The security parameter is κ.

The protocol:

1. **Preliminary phase:**
   (a) Let $\hat{y}$ be some default input value for P_2 and $\hat{x}$ some default input for P_1. We define $a_0 = f_1(x, \hat{y})$ and $b_0 = f_2(\hat{x}, y)$.
   (b) Parties P_1 and P_2 run some (unfair) protocol π for computing ShareGen, using their respective inputs x, y, and security parameter κ.
   (c) If P_1 receives $\perp$ from the above computation (because P_2 aborts the computation or uses an invalid input in π) she outputs a_0 and halts. Likewise, if P_2 receives $\perp$, he outputs b_0 and halts. Otherwise, the parties proceed.
   (d) Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and k_a.
   (e) Denote the output of P_2 from π by $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, $b_1^{(2)}, \ldots, b_m^{(2)}$, and k_b.
2. **For $i = 1, \ldots, m$ do:**
   P_2 sends the next share to P_1:
   (a) P_2 sends $(a_i^{(2)}, t_i^a)$ to P_1.
   (b) P_1 receives $(a_i^{(2)}, t_i^a)$ from P_2. If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 0$ (or if P_1 received an invalid message, or no message), then P_1 outputs a_{i-1} and halts
   (c) If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 1$, P_1 computes and stores the value $a_i = a_i^{(1)} \oplus a_i^{(2)}$ and continues running the protocol.
   P_1 sends the next share to P_2:
   (a) P_1 sends $(b_i^{(1)}, t_i^b)$ to P_2.
   (b) P_2 receives $(b_i^{(1)}, t_i^b)$ from P_1. If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 0$ (or if P_2 received an invalid message, or no message), then P_2 outputs b_{i-1} and halts.
   (c) If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 1$, P_2 computes and stores the value $b_i = b_i^{(1)} \oplus b_i^{(2)}$ and continues running the protocol.

Outputs: P_1 outputs a_m and P_2 outputs b_m.

Figure 2.2: Protocol for computing $\mathcal{F}$, using ShareGen.

Once the players have each obtained their two sequences of secret shares, they alternate exchanging them for r rounds. Player two begins round i by sending $a_i^{(2)}$, enabling player one to compute $a_i = a_i^{(1)} \oplus a_i^{(2)}$. Player one then responds by sending $b_i^{(1)}$, which enables player two to compute $b_i = b_i^{(1)} \oplus b_i^{(2)}$. We note that the way the values $(a_1, \ldots, a_r)$ and $(b_1, \ldots, b_r)$ are chosen will depend on the particular function $\mathcal{F}$ being computed. If some player aborts in round i, the other player simply outputs the value they computed in the prior round: e.g., if player two fails to send $a_i^{(2)}$, then player one outputs a_{i-1} and halts. The formal description of how this is done is presented in Figure 2.2. We also use message authentication codes (MACs) to prevent a player from sending an incorrect share; if a player sends a share that does not correctly verify, this is treated as an abort. Effectively, then, the only option for a malicious player is to play honestly, or choose to abort.

Note that the format is quite general. Although we will not prove this formally, it is not too hard to see that any two-party protocol π for computing $\mathcal{F}$ can be compiled into a protocol

with the above format. Intuitively, we simply view ShareGen as simulating the protocol "in its head", determining the outputs of each player at each round should their opponent abort, and setting the values of a_i and b_i to be exactly these output values.

Chapter 3

Complete Fairness in Secure Two-Party Computation

As we have discussed in Chapter 1, it has been known since the early 1980's that fairly computing certain functions is impossible, including the natural example of signature exchange, and even the far simpler function of bit exchange. Although the security model at that time was not yet standardized, it is easy to see that the impossibility results extend to the definition of fairness implied by Defintion 2.3.1. In this chapter, we demonstrate several surprising results that show how to compute certain interesting functions with complete fairness according to Definition 2.3.1. We begin with a fairly simple protocol for the Millionaire's problem, and then proceed to give a slightly more involved protocol that computes a wider class of functions. The work in this Chapter first appeared at STOC 2008 [49].

3.1 Fair Computation of the Millionaires' Problem (and More)

In this section, we describe a protocol for securely computing the millionaires' problem (and related functionalities) with complete fairness. Specifically, we look at functions defined by a lower-triangular matrix, as in the following table:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	0	0	0	0	0	0
x_2	1	0	0	0	0	0
x_3	1	1	0	0	0	0
x_4	1	1	1	0	0	0
x_5	1	1	1	1	0	0
x_6	1	1	1	1	1	0

Let $\mathcal{F} = \{f_{m(\kappa)}\}_{\kappa \in \mathbb{N}}$ denote a function of the above form, where $m = m(\kappa)$ denotes the size of the domain of each input which we assume have the same size. (Handling the case where they are unequal involves a trivial technical change to the protocol. The details are described in the original work [49].) Let $X_m = \{x_1, \ldots, x_m\}$ denote the valid inputs for the first party and let $Y_m = \{y_1, \ldots, y_m\}$ denote the valid inputs for the second party. By suitably ordering these elements, we may write f_m as follows:

$$f_m(x_i, y_j) = \begin{cases} 1 & \text{if } i > j \\ 0 & \text{if } i \leq j \end{cases} . \tag{3.1}$$

Viewed in this way, f_m is exactly the millionaires' problem or, equivalently, the "greater than" function. The remainder of this section is devoted to a proof of the following theorem:

Theorem *Let $m = \mathsf{poly}(\kappa)$. Assuming the existence of constant-round general secure two-party computation with abort, there exists an $\Theta(m)$-round protocol that securely computes $\mathcal{F} = \{f_m\}$ with complete fairness.*

Constant-round protocols for general secure two-party computation with abort can be constructed based on enhanced trapdoor permutations or any constant-round protocol for oblivious transfer [68]. (The assumption of a constant-round protocol is only needed for the claim regarding round complexity.) The fact that our protocol requires (at least) $\Theta(m)$ rounds explains why we require $m = \mathsf{poly}(\kappa)$. When $m = 2$, we obtain a constant-round protocol for computing boolean AND with complete fairness and, by symmetry, we also obtain a protocol for boolean OR. We remark further that our results extend to variants of f_m such as the "greater than or equal to" function, or the "greater than" function where the sizes of the domains X and Y are unequal, and most generally, to any function without an "embedded XOR" (defined in Section 3.2); see [49] for a full discussion.

3.1.1 The Protocol

In this section, we write f in place of f_m, and X and Y in place of X_m and Y_m.

Intuition. At a high level, our protocol works as follows. Say the input of P_1 is x_i, and the input of P_2 is y_j. Following a constant-round "pre-processing" phase, the protocol proceeds in a series of m iterations, where P_1 learns the output — namely, the value $f(x_i, y_j)$ — in iteration i, and P_2 learns the output in iteration j. (That is, in contrast to standard protocols, the iteration in which a party learns the output *depends on the value of its own input*.) If one party (say, P_1) aborts after receiving its iteration-k message, and the second party (say, P_2) has not yet received its output, then P_2 "assumes" that P_1 learned its output in iteration k, and computes f on its own using input x_k for P_1. (In this case, that means that P_2 would output $f(x_k, y_j)$.) We stress that a malicious P_1 may, of course, abort in any iteration it likes (and not necessarily in the iteration in which it learns its output); the foregoing is only an intuitive explanation.

The key observation for why this yields a fair protocol is as follows. If P_1 aborts before he learns anything then the protocol is still fair. On the other hand, if he aborts after he has learned the output, then he has also revealed an upper bound on his input! We therefore assume that he has learned something, and use the information that this provides to ensure fairness. More specifically, say P_1 is malicious and uses x_i as its effective input. Let y_j denote the input of P_2. There are two possibilities: P_1 either aborts in iteration $k < i$, or iteration $k \geq i$. (If P_1 never aborts then fairness is trivially achieved.) In the first case, P_1 never learns the correct output and so fairness is achieved. In the second case, P_1 does obtain the output $f(x_i, y_j)$ (in iteration i) and then aborts in some iteration $k \geq i$. Here we consider two subcases depending on the value of P_2's input y_j:

- If $j < k$ then P_2 has already received its output in a previous iteration and fairness is achieved.
- If $j \geq k$ then P_2 has not yet received its output. Since P_1 aborts in iteration k, the protocol directs P_2 to output $f(x_k, y_j)$. However, since $j \geq k \geq i$, we have $f(x_k, y_j) = 0 = f(x_i, y_j)$ (relying on the specifics of f), and so the output of P_2 is equal to the output

obtained by P_1 (and thus fairness is achieved). This is the key observation that enables us to obtain fairness for this function.

We formalize the above intuition in our proof, where we demonstrate an ideal-world simulator corresponding to the actions of any malicious P_1. Of course, we also consider the case of a malicious P_2.

Formal description of the protocol. Technically, the protocol proceeds according to the template described in Section 2.8. All we need to do is to specify how the values $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ are set within ShareGen. (Recall, this is the only component of the canonical form that is function specific.) To aid the reader, however, we also include an outline of the complete protocol here, ignoring some of the details that were elaborated in Section 2.8 (such as the message authentication). Recall that the parties first execute ShareGen(x_i, y_j) in order to receive secret shares of the values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$. We set:

$$a_k = \begin{cases} f(x_i, y_k) & \text{if } k < i \\ f(x_i, y_j) & \text{if } k \geq i \end{cases}$$

$$b_k = \begin{cases} f(x_{k+1}, y_j) & \text{if } k < j \\ f(x_i, y_j) & \text{if } k \geq j \end{cases}$$

The players then exchange their secret shares one-by-one in a sequence of m iterations. Specifically, in iteration i party P_2 will send $a_i^{(2)}$ to P_1, thus allowing P_1 to reconstruct the value $a_i \stackrel{\text{def}}{=} a_i^{(1)} \oplus a_i^{(2)}$, and then P_1 will send $b_i^{(1)}$ to P_2, thus allowing P_2 to learn the value $b_i \stackrel{\text{def}}{=} b_i^{(2)} \oplus b_i^{(1)}$.

ShareGen

Inputs: Let the inputs to ShareGen be x_i and y_j. If one of the received inputs is not in the correct domain, then both parties are given output $\perp$. The security parameter is κ.

Computation:

1. Define values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$ in the following way:

$$a_k = \begin{cases} f(x_i, y_k) & \text{if } i > k \\ f(x_i, y_j) & \text{if } i \leq k \end{cases}$$

$$b_k = \begin{cases} f(x_{k+1}, y_j) & \text{if } j > k \\ f(x_i, y_j) & \text{if } j \leq k \end{cases}$$

2. For $1 \leq k \leq m$, choose $(a_k^{(1)}, a_k^{(2)})$ and $(b_k^{(1)}, b_k^{(2)})$ as random secret sharings of a_k and b_k, respectively. (I.e., $a_k^{(1)}$ is random and $a_k^{(1)} \oplus a_k^{(2)} = a_k$.)

Output:

1. P_1 receives the values $a_1^{(1)}, \ldots, a_m^{(1)}, b_1^{(1)}, \ldots, b_m^{(1)}$.
2. P_2 receives the values $a_1^{(2)}, \ldots, a_m^{(2)}, b_1^{(2)}, \ldots, b_m^{(2)}$.

Figure 3.1: Functionality ShareGen.

Theorem 3.1.1 *If* (Gen, Mac, Vrfy) *is an information-theoretically secure* m*-time MAC, and* π *securely computes* ShareGen *with abort, then the protocol in Figure 3.2 securely computes* $\{f_m\}$ *with complete fairness.*

Π_{mill}

Inputs: Party P_1 has input x_i and party P_2 has input y_j. The security parameter is κ.

The protocol:

1. **Preliminary phase:**
   (a) We define $a_0 = f(x_i, y_0)$ and $b_0 = f(x_1, y_j)$.
   (b) Parties P_1 and P_2 run some (unfair) protocol π for computing ShareGen, using their respective inputs x_i, y_j, and security parameter κ.
   (c) If P_1 receives $\perp$ from the above computation she outputs a_0 and halts. Likewise, if P_2 receives $\perp$, he outputs b_0 and halts. Otherwise, the parties proceed.
   (d) Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_m^{(1)}, b_1^{(1)}, \ldots, b_m^{(1)}$.
   (e) Denote the output of P_2 from π by $a_1^{(2)}, \ldots, a_m^{(2)}, b_1^{(2)}, \ldots, b_m^{(2)}$.
2. **For $i = 1, \ldots, m$ do:**
   P_2 sends the next share to P_1:
   (a) P_2 sends $a_i^{(2)}$ to P_1.
   (b) If P_1 received an invalid message, or no message, then P_1 outputs a_{i-1} and halts.
   P_1 sends the next share to P_2:
   (a) P_1 sends $b_i^{(1)}$ to P_2.
   (b) If P_2 received an invalid message, or no message, then P_2 outputs b_{i-1} and halts.

Outputs: P_1 outputs a_m and P_2 outputs b_m.

Figure 3.2: Protocol for computing the Millionaire problem, using ShareGen.

Proof: Let Π_{mill} denote the protocol in Figure 3.2. We analyze Π_{mill} in a hybrid model where there is a trusted party computing ShareGen. (Note: since π is only guaranteed to securely compute ShareGen with abort, the adversary in the hybrid model is allowed to abort the trusted party computing ShareGen before output is sent to the honest party.) We prove that an execution of Π_{mill} in this hybrid model is *statistically close* to an evaluation of f in the ideal model (with complete fairness), where the only difference occurs due to MAC forgeries. Applying Proposition 2.7.1 then implies the theorem.

We separately analyze corruption of P_1 and P_2, beginning with P_1:

Claim 3.1.2 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_1 and running Π_{mill} in a hybrid model with access to an ideal functionality computing* ShareGen *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*,\kappa\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}}_{\Pi_{\mathsf{mill}},\mathcal{A}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*,\kappa\in\mathbb{N}}$$

Proof: Let P_1 be corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input x, the auxiliary input z, and the security parameter κ.
2. $\mathcal{S}$ receives the input x' of $\mathcal{A}$ to the computation of the functionality ShareGen.
   (a) If $x' \notin X$ (this includes the case when $x' = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen, sends x_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.

(b) Otherwise, if the input is some $x' \in X$, then $\mathcal{S}$ chooses uniformly-distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. In addition, it generates keys $k_a, k_b \leftarrow \mathsf{Gen}(1^\kappa)$ and computes $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$ for every i. Finally, it hands $\mathcal{A}$ the strings $a_1^{(1)}, \ldots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and k_a as its output from the computation of $\mathsf{ShareGen}$.

3. If $\mathcal{A}$ sends abort to the trusted party computing $\mathsf{ShareGen}$ (signalling that P_2 should receive $\perp$ as output from $\mathsf{ShareGen}$), then $\mathcal{S}$ sends x_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.
4. Let i (with $1 \leq i \leq m$) be the index such that $x' = x_i$ (such an i exists since $x' \in X$).
5. To simulate iteration j, for $j < i$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ chooses $a_j^{(2)}$ such that $a_j^{(1)} \oplus a_j^{(2)} = f(x_i, y_j) = 1$, and computes the tag $t_j^a = \mathsf{Mac}_{k_a}(j \| a_j^{(2)})$. Then $\mathcal{S}$ gives $\mathcal{A}$ the message $(a_j^{(2)}, t_j^a)$.
   (b) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{b}_j^{(1)}, \hat{t}_j^b)$ in the jth iteration:
      i. If $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends x_j to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.
      ii. If $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 1$, then $\mathcal{S}$ proceeds to the next iteration.
6. To simulate iteration i, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ sends x_i to the trusted party computing f, and receives back the output $z = f(x_i, y)$.
   (b) $\mathcal{S}$ chooses $a_i^{(2)}$ such that $a_i^{(1)} \oplus a_i^{(2)} = z$, and computes the tag $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$. Then $\mathcal{S}$ gives $\mathcal{A}$ the message $(a_i^{(2)}, t_i^a)$.
   (c) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{b}_i^{(1)}, \hat{t}_i^b)$. If $\mathsf{Vrfy}_{k_b}(i \| \hat{b}_i^{(1)}, \hat{t}_i^b) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 1$, then $\mathcal{S}$ proceeds to the next iteration.
7. To simulate iteration j, for $i < j \leq m$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ chooses $a_j^{(2)}$ such that $a_j^{(1)} \oplus a_j^{(2)} = z$, and computes the tag $t_j^a = \mathsf{Mac}_{k_a}(j \| a_j^{(2)})$. Then $\mathcal{S}$ gives $\mathcal{A}$ the message $(a_j^{(2)}, t_j^a)$.
   (b) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{b}_j^{(1)}, \hat{t}_j^b)$. If $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 1$, then $\mathcal{S}$ proceeds to the next iteration.
8. If $\mathcal{S}$ has not halted yet, at this point it halts and outputs whatever $\mathcal{A}$ outputs.

We analyze the simulator $\mathcal{S}$ described above. In what follows we assume that if $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 1$ then $\hat{b}_j^{(1)} = b_j^{(1)}$ (meaning that $\mathcal{A}$ sent the same share that it received).

Under this assumption, we show that the distribution generated by $\mathcal{S}$ is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_2. Since this assumption holds with all but negligible probability (by security of the information-theoretic MAC), this proves statistical closeness as stated in the claim.

Let y denote the input of P_2. It is clear that the view of $\mathcal{A}$ in an execution with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ in a hybrid execution with P_2; the only difference is that the initial shares given to $\mathcal{A}$ are generated by $\mathcal{S}$ without knowledge of $z = f(x', y)$, but since these shares are uniformly distributed the view of $\mathcal{A}$ is unaffected. Therefore, what is left to demonstrate is that the joint distribution of $\mathcal{A}$'s view and P_2's output is identical in the hybrid world and the ideal world. We show this now by separately considering three different cases:

1. *Case 1: $\mathcal{S}$ sends x_1 to the trusted party because $x' \notin X$, or because $\mathcal{A}$ aborted the computation of* ShareGen*:* In the hybrid world, P_2 would have received $\perp$ from ShareGen, and would have then output $f(x_1, y)$ as instructed by protocol Π_{mill}. This is exactly what P_2 outputs in the ideal execution with $\mathcal{S}$ because, in this case, $\mathcal{S}$ sends x_1 to the trusted party computing f.

   If Case 1 does not occur, let x_i be defined as in the description of the simulator.

2. *Case 2: $\mathcal{S}$ sends x_j to the trusted party, for some $j < i$:* This case occurs when $\mathcal{A}$ aborts the protocol in some iteration $j < i$ (either by refusing to send a message, sending an invalid message, or sending an incorrect share). There are two sub-cases depending on the value of P_2's input y. Let ℓ be the index such that $y = y_\ell$. Then:

   (a) If $\ell \geq j$ then, in the hybrid world, P_2 would not yet have determined its output (since it only determines its output once it receives a valid message from P_1 in iteration ℓ). Thus, as instructed by the protocol, P_2 would output $f(x_j, y)$. This is exactly what P_2 outputs in the ideal world, because $\mathcal{S}$ sends x_j to the trusted party in this case.

   (b) If $\ell < j$ then, in the hybrid world, P_2 would have already determined its output $f(x', y) = f(x_i, y_\ell)$ in the ℓth iteration. In the ideal world, P_2 will output $f(x_j, y_\ell)$ since $\mathcal{S}$ sends x_j to the trusted party. Since $j < i$ we have $\ell < j < i$ and so $f(x_j, y_\ell) = f(x_i, y_\ell) = 1$. Thus, P_2's output $f(x_i, y)$ in the hybrid world is equal to its output $f(x_j, y)$ in the ideal execution with $\mathcal{S}$.

3. *Case 3: $\mathcal{S}$ sends x_i to the trusted party:* Here, P_2 outputs $f(x_i, y)$ in the ideal execution. We show that this is identical to what P_2 would have output in the hybrid world. There are two sub-cases depending on P_2's input y. Let ℓ be the index such that $y = y_\ell$. Then:

   (a) If $\ell < i$, then P_2 would have already determined its output $f(x', y) = f(x_i, y)$ in the ℓth iteration. (The fact that we are in Case 3 means that $\mathcal{A}$ could not have sent an incorrect share prior to iteration i.)

   (b) If $\ell \geq i$, then P_2 would not yet have determined its output. There are two sub-cases:

      i. $\mathcal{A}$ sends correct shares in iterations $j = i, \ldots, \ell$ (inclusive). Then P_2 would determine its output as $b_\ell^{(1)} \oplus b_\ell^{(2)} = f(x', y) = f(x_i, y)$, exactly as in the ideal world.

      ii. $\mathcal{A}$ sends an incorrect share in iteration ζ, where $i \leq \zeta \leq \ell$. In this case, by the specification of the protocol, party P_2 would output $f(x_\zeta, y) = f(x_\zeta, y_\ell)$. However, since $i \leq \zeta \leq \ell$ we have $f(x_\zeta, y_\ell) = 0 = f(x_i, y_\ell)$. Thus, P_2 outputs the same value in the hybrid and ideal executions.

This concludes the proof of the claim. ■

The following claim, dealing with a corrupted P_2, completes the proof of the theorem:

Claim 3.1.3 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_2 and running Π_{mill} in a hybrid model with access to an ideal functionality computing* ShareGen *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_2 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*,\kappa\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}}_{\Pi_{\mathsf{mill}},\mathcal{A}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*,\kappa\in\mathbb{N}}.$$

Proof: Say P_2 is corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input y, the auxiliary input z, and the security parameter κ.
2. $\mathcal{S}$ receives the input y' of $\mathcal{A}$ to the computation of the functionality ShareGen.
   (a) If $y' \notin Y$ (this includes the case when $y' = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen, sends y_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.
   (b) Otherwise, if the input is some $y' \in Y$, then $\mathcal{S}$ chooses uniformly-distributed shares $a_1^{(2)},\ldots,a_m^{(2)}$ and $b_1^{(2)},\ldots,b_m^{(2)}$. In addition, it generates keys $k_a, k_b \leftarrow \mathsf{Gen}(1^\kappa)$ and computes $t_i^a = \mathsf{Mac}_{k_a}(i\|a_i^{(2)})$ for every i. Finally, it hands $\mathcal{A}$ the strings $b_1^{(2)},\ldots,b_m^{(2)}$, $(a_1^{(2)},t_1^a),\ldots,(a_m^{(2)},t_m^a)$, and k_b as its output from the computation of ShareGen.
3. If $\mathcal{A}$ sends abort to the trusted party computing ShareGen, then $\mathcal{S}$ sends y_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.
4. Let i (with $1 \le i \le m$) be the index such that $y' = y_i$ (such an i exists since $y' \in Y$).
5. To simulate iteration j, for $j < i$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_j^{(2)},\hat{t}_j^a)$ in the jth iteration:
      i. If $\mathsf{Vrfy}_{k_a}(j\|\hat{a}_j^{(2)},\hat{t}_j^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends y_{j-1} to the trusted party computing f (if $j = 1$, then $\mathcal{S}$ sends y_1), outputs whatever $\mathcal{A}$ outputs, and halts.
      ii. If $\mathsf{Vrfy}_{k_a}(j\|\hat{a}_j^{(2)},\hat{t}_j^a) = 1$, then $\mathcal{S}$ proceeds.
   (b) Choose $b_j^{(1)}$ such that $b_j^{(1)}\oplus b_j^{(2)} = f(x_{j+1},y_i)$, and compute the tag $t_j^b = \mathsf{Mac}_{k_b}(j\|b_j^{(1)})$. Then give to $\mathcal{A}$ the message $(b_j^{(1)},t_j^b)$.
6. To simulate iteration i, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_i^{(2)},\hat{t}_i^a)$.
      i. If $\mathsf{Vrfy}_{k_a}(i\|\hat{a}_i^{(2)},\hat{t}_i^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends y_{i-1} to the trusted party computing f (if $i = 1$ then $\mathcal{S}$ sends y_1), outputs whatever $\mathcal{A}$ outputs, and halts.
      ii. If $\mathsf{Vrfy}_{k_a}(i\|\hat{a}_i^{(2)},\hat{t}_i^a) = 1$, then $\mathcal{S}$ sends y_i to the trusted party computing f, receives the output $z = f(x,y_i)$, and proceeds.
   (b) Choose $b_i^{(1)}$ such that $b_i^{(1)}\oplus b_i^{(2)} = z$, and compute the tag $t_i^b = \mathsf{Mac}_{k_b}(i\|b_i^{(1)})$. Then give to $\mathcal{A}$ the message $(b_i^{(1)},t_i^b)$.

7. To simulate iteration j, for $i < j \leq m$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_j^{(2)}, \hat{t}_j^a)$. If $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 1$, then $\mathcal{S}$ proceeds.
   (b) Choose $b_j^{(1)}$ such that $b_j^{(1)} \oplus b_j^{(2)} = z$, and compute the tag $t_j^b = \mathsf{Mac}_{k_b}(j \| b_j^{(1)})$. Then give to $\mathcal{A}$ the message $(b_j^{(1)}, t_j^b)$.
8. If $\mathcal{S}$ has not halted yet, at this point it halts and outputs whatever $\mathcal{A}$ outputs.

As in the proof of the previous claim, we assume in what follows that if $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 1$ then $\hat{a}_j^{(2)} = a_j^{(2)}$ (meaning that $\mathcal{A}$ sent P_1 the same share that it received). Under this assumption, we show that the distribution generated by $\mathcal{S}$ is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_1. Since this assumptions holds with all but negligible probability (by security of the MAC), this proves statistical closeness as stated in the claim.

Let x denote the input of P_1. Again, it is clear that the view of $\mathcal{A}$ in an execution with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ in a hybrid execution with P_1. What is left to demonstrate is that the joint distribution of $\mathcal{A}$'s view and P_1's output is identical. We show this by considering four different cases:

1. *Case 1: $\mathcal{S}$ sends y_1 to the trusted party because $y' \notin Y$, or because $\mathcal{A}$ aborted the computation of* ShareGen*:* In such a case, the protocol instructs P_1 to output $f(x, y_1)$, exactly what P_1 outputs in the ideal world.

2. *Case 2: $\mathcal{S}$ sends y_1 to the trusted party because $\mathcal{A}$ sends an incorrect share in the first iteration:* In this case, the simulator sends y_1 to the trusted party computing f, and so the output of P_1 in the ideal world is $f(x, y_1)$. In the hybrid world, P_1 will also output $f(x, y_1)$ as instructed by the protocol.

   If Cases 1 and 2 do not occur, let y_i be defined as in the description of the simulator.

3. *Case 3: $\mathcal{S}$ sends y_{j-1} to the trusted party, for some $1 \leq j-1 < i$, because $\mathcal{A}$ sends an incorrect share in the jth iteration:*

   The output of P_1 in the ideal world is $f(x, y_{j-1})$. There are two sub-cases here, depending on the value of P_1's input x. Let ℓ be the index such that $x = x_\ell$. Then:

   (a) If $\ell < j$ then, in the hybrid world, P_1 would have already determined its output $f(x, y') = f(x_\ell, y_i)$. But since $\ell \leq j - 1 < i$ we have $f(x_\ell, y_i) = 0 = f(x_\ell, y_{j-1})$, and so P_1's output is identical in both the hybrid and ideal worlds.
   (b) If $\ell \geq j$ then, in the hybrid world, P_1 would not yet have determined its output. Therefore, as instructed by the protocol, P_1 will output $f(x, y_{j-1})$ in the hybrid world, which is exactly what it outputs in the ideal execution with $\mathcal{S}$.

4. *Case 4: $\mathcal{S}$ sends y_i to the trusted party:* This case occurs when $\mathcal{A}$ sends correct shares up through and including iteration i. The output of P_1 in the ideal world is $f(x, y_i)$. There are again two sub-cases here, depending on the value of P_1's input x. Let ℓ be the index such that $x = x_\ell$. Then:

   (a) If $\ell \leq i$, then P_1 would have already determined its output $f(x, y') = f(x_\ell, y_i)$ in the ℓth iteration. This matches what P_1 outputs in the ideal execution with $\mathcal{S}$.

(b) If $\ell > i$, then P_1 would not have yet have determined its output. There are two sub-cases:

i. $\mathcal{A}$ sends correct shares in iterations $j = i+1, \ldots, \ell$ (inclusive). This implies that, in the hybrid world, P_1 would determine its output to be $a_\ell^{(1)} \oplus a_\ell^{(2)} = f(x, y') = f(x, y_i)$, exactly as in the ideal execution.

ii. $\mathcal{A}$ sends an incorrect share in iteration ζ, where $i < \zeta \leq \ell$. In this case, by the specification of the protocol, party P_1 would output $f(x, y_{\zeta-1}) = f(x_\ell, y_{\zeta-1})$ in the hybrid world. But since $i \leq \zeta - 1 < \ell$ we have $f(x_\ell, y_{\zeta-1}) = 1 = f(x_\ell, y_i)$, and so P_1's output is identical in both the hybrid and ideal worlds.

This completes the proof of the claim. ■

The preceding claims along with Proposition 2.7.1 imply the theorem. ■

3.2 Fair Computation of Functions with an Embedded XOR

We say that a function $\mathcal{F}$ has an embedded XOR if there exist inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$. We prove in [49] that the protocol presented in the preceding section can be used to compute any boolean function without an embedded XOR, providing complete security and fairness. Essentially, it is not hard to see that any such function can be "rearranged" to fit the description of the greater than function given in Section 3.1.

Recall that Cleve's result showing impossibility of completely-fair coin tossing implies the impossibility of completely-fair computation of boolean XOR. More generally, it implies the impossibility of completely-fair computation of any function f that *enables* coin tossing. Furthermore, note that the protocol from the previous section relies heavily on the property that function does not contain an embedded XOR; specifically, we require that for $j \geq k \geq i$, we have $f(x_k, y_j) = f(x_i, y_j)$ and this property does not hold if y_j is part of an embedded XOR with x_k and x_i, regardless of how we order the inputs. It is tempting to conclude, therefore, that no function containing an embedded XOR can be computed with complete fairness. In this section, we show that this is not the case — that there exist functions with an embedded XOR that *can* be computed with complete fairness. Interestingly, however, such functions appear to be "more difficult" to compute with complete fairness; specifically, we refer the reader to Section 3.3 where we prove a lower bound of $\omega(\log \kappa)$ on the round complexity of any protocol for completely-fair computation of any function having an embedded XOR. (Note that, in general, this bound is incomparable to the result of the previous section, where the round complexity was linear in the domain size.)

It will be instructive to see why Cleve's impossibility result does not immediately rule out complete fairness for all functions containing an embedded XOR. Consider the following function f (which is the example for which we will later prove feasibility):

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

If the parties could be forced to choose their inputs from $\{x_1, x_2\}$ and $\{y_1, y_2\}$, respectively, then it would be easy to generate a fair coin toss from any secure computation of f (with complete fairness) by simply instructing both parties to choose their inputs uniformly from the stated domains. (This results in a fair coin toss since the output is uniform as long as

either party chooses their input at random.) Unfortunately, a protocol for securely computing f does *not* restrict the first party to choosing its input in $\{x_1, x_2\}$, and cannot prevent that party from choosing input x_3 and thus biasing the result toward 1 with certainty. (Naive solutions, such as modifying the protocol to require the first party to provide a zero-knowledge proof that it chose its input in $\{x_1, x_2\}$, do not work. The reason is that we still need a way for the second party to decide on their output in case the zero-knowledge proof of the first party fails. The original fair protocol for computing f may specify (and rely upon the fact) that P_2 with input y should default to outputting $f(x_3, y)$ when P_1 cheats.)

Of course, this only shows that Cleve's impossibility result does not apply, but it does not prove that a completely-fair protocol for computing f exists. In this section we present a generic protocol for computing a boolean function $\mathcal{F} = \{f_\kappa : X_\kappa \times Y_\kappa \to \{0,1\}\}$. (For convenience, we write X and Y and drop the explicit dependence on κ in what follows.) The protocol is parameterized by a function $\alpha = \alpha(\kappa)$, and the number of rounds is set to $m = \omega(\alpha^{-1} \log \kappa)$ in order for correctness to hold with all but negligible probability. (We thus must have α noticeable to ensure that the number of rounds is polynomial in κ.)

We do *not* claim that the protocol is completely-fair for arbitrary functions $\mathcal{F}$ and arbitrary settings of α. Rather, we claim that for *some* functions $\mathcal{F}$ there exists a corresponding α for which the protocol is completely fair. In Section 3.2.1, we prove this for one specific function that contains an embedded XOR. In Section 3.2.2 we generalize the proof and show that the protocol can be used for completely-fair computation of a wider class of functions. We categorize the class of functions there.

Overview and intuition. As before, the parties begin by executing an unfair computation of ShareGen (cf. Figure 3.3) and receive as output secret shares of $a_1, b_1, \ldots, a_m, b_m$. As always, these values are generated based on the parties' respective inputs x and y, and it remains only to describe how these values are fixed.

In contrast to our earlier protocol, however, the values $a_1, b_1, \ldots, a_m, b_m$ are now generated *probabilistically* in the following way: first, a value $i^* \in \{1, \ldots, m\}$ is chosen according to a geometric distribution with parameter α (see below). For $i < i^*$, the value a_i (resp., b_i) is chosen in a manner that is *independent* of P_2's (resp., P_1's) input; specifically, we set $a_i = f(x, \hat{y})$ for randomly-chosen $\hat{y} \in Y$ (and analogously for b_i). For all $i \geq i^*$, the values a_i and b_i are set equal to $f(x, y)$. More formally,

$$a_i = \begin{cases} f(x, \hat{y}) & \text{if } i < i^* \mid \text{where } \hat{y} \xleftarrow{r} Y \\ f(x, y) & \text{if } i \geq i^* \end{cases}$$

$$b_i = \begin{cases} f(\hat{x}, y) & \text{if } i < i^* \mid \text{where } \hat{x} \xleftarrow{r} X \\ f(x, y) & \text{if } i \geq i^* \end{cases}$$

We note that fresh randomness is used in selecting $\hat{x}$ and $\hat{y}$ each time. Note that if $m = \omega(\log \kappa)$, we have $a_m = b_m = f(x, y)$ with all but negligible probability and so correctness holds. (The protocol could also be modified so that $a_m = b_m = f(x, y)$ with probability 1, thus giving perfect correctness. But the analysis is easier without this modification.)

Fairness is more difficult to see and, of course, cannot hold for all functions f, since some functions cannot be computed fairly. But as intuition for why the protocol achieves fairness for certain functions, we observe that: (1) if a malicious party (say, P_1) aborts in some iteration $i < i^*$, then P_1 has not yet obtained any information about P_2's input and so fairness is trivially achieved. On the other hand, (2) if P_1 aborts in some iteration $i > i^*$ then *both* P_1 and P_2 have received the correct output $f(x, y)$ and fairness is obtained. The worst case,

ShareGen

Inputs: Let the inputs to ShareGen be $x \in X$ and $y \in Y$. (If one of the received inputs is not in the correct domain, then both parties are given output $\perp$.)

Computation:

1. Define values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$ in the following way:
   - Choose i^* according to a geometric distribution with parameter α (see text).
   - For $i = 1$ to $i^* - 1$ do:
     - Choose $\hat{y} \leftarrow Y$ and set $a_i = f(x, \hat{y})$.
     - Choose $\hat{x} \leftarrow X$ and set $b_i = f(\hat{x}, y)$.
   - For $i = i^*$ to m, set $a_i = b_i = f(x, y)$.
2. For $1 \leq i \leq m$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (E.g., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)

Output:

1. Send to P_1 the values $(a_1^{(1)}, \ldots, a_m^{(1)}, b_1^{(1)}, \ldots, b_m^{(1)})$
2. Send to P_2 the values $(a_1^{(2)}, \ldots, a_m^{(2)}, b_1^{(2)}, \ldots, b_m^{(2)})$

Figure 3.3: Functionality ShareGen for functions with embedded XOR.

then, occurs when P_1 aborts exactly in iteration i^*, as it has then learned the correct value of $f(x, y)$ while P_2 has not (since P_2 has only seen values $b_1, \ldots b_{i^*-1}$, which are independent of P_1's input). However, P_1 cannot identify iteration i^* with certainty (this holds even if it knows the other party's input y) and, even though it may guess i^* correctly with noticeable probability, the fact that it can never be sure whether its guess is correct will suffice to ensure fairness. Recall that fairness is defined only through an indistinguishability requirement; if P_1 outputs the correct output while P_2 does not, this fact by itself does not violate fairness.[1] This point will play a key role when we prove security: the simulator will rely on the fact that any malicious P_1 that aborts in round i^* *must* also, with noticeable probability, abort in some round $i < i^*$. Furthermore, we will use the fact that one case is indistinguishable from the other.

Formal description of the protocol. The protocol is parameterized by a value $\alpha = \alpha(\kappa)$ which is assumed to be noticeable. Let $m = \omega(\alpha^{-1} \log \kappa)$. As usual, we use an m-time MAC with information-theoretic security, however to ease the reading we ignore these details and refer the reader to Section 2.8 for technical details.

As described above, ShareGen generates a value i^* according to a *geometric distribution* with parameter α. This is the probability distribution on $\mathbb{N} = \{1, 2, \ldots\}$ given by repeating a Bernoulli trial (with parameter α) until the first success. In other words, i^* is determined by tossing a biased coin (that is heads with probability α) until the first head appears, and letting i^* be the number of tosses performed. We remark that, as far as ShareGen is concerned, if $i^* > m$ then the exact value of i^* is unimportant, and so ShareGen can be implemented in strict (rather than expected) polynomial time. In any case, our choice of m ensures that $i^* \leq m$ with all but negligible probability.

Again, as before, the protocol for computing f begins by calling ShareGen as a subroutine

[1] If this is unsettling, consider a different adversary that aborts the protocol before it begins, and attempts to guess the output. This adversary may also output the correct value alone, but intuitively, there is nothing unfair in his action.

and then has the parties exchange their shares. A formal description of the protocol is given in Figure 3.4. Note that the players do not know the value of i^*, as this was selected inside the (secure) randomized protocol for ShareGen.

Π_{EXOR}

Inputs: Party P_1 has input x and party P_2 has input y. The security parameter is κ.

The protocol:

1. **Preliminary phase:**
   (a) P_1 chooses $\hat{y} \in Y$ uniformly at random, and sets $a_0 = f(x, \hat{y})$. Similarly, P_2 chooses $x \in X$ uniformly at random, and sets $b_0 = f(\hat{x}, y)$.
   (b) Parties P_1 and P_2 run protocol π for computing ShareGen, using their respective inputs x and y, and security parameter κ.
   (c) If P_1 receives $\perp$ from the above computation, it outputs a_0 and halts. Likewise, if P_2 receives $\perp$ then it outputs b_0 and halts. Otherwise, the parties proceed to the next step.
   (d) Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_m^{(1)}, b_1^{(1)}, \ldots, b_m^{(1)}$.
   (e) Denote the output of P_2 from π by $a_1^{(2)}, \ldots, a_m^{(2)}, b_1^{(2)}, \ldots, b_m^{(2)}$.
2. **For $i = 1, \ldots, m$ do:**
   P_2 sends the next share to P_1:
   (a) P_2 sends $a_i^{(2)}$ to P_1.
   (b) P_1 receives $a_i^{(2)}$ from P_2. If P_1 received an invalid message, or no message, then P_1 outputs a_{i-1} and halts.
   P_1 sends the next share to P_2:
   (a) P_1 sends $b_i^{(1)}$ to P_2.
   (b) P_2 receives $b_i^{(1)}$ from P_1. If P_2 received an invalid message, or no message), then P_2 outputs b_{i-1} and halts.

Outputs: P_1 outputs a_m and P_2 outputs b_m.

Figure 3.4: Protocol for computing a function f.

3.2.1 Proof of Security for a Particular Function

Protocol Π_{EXOR} cannot guarantee complete fairness for *all* functions f. Rather, what we claim is that for *certain* functions f and particular associated values of α, the protocol provides complete fairness. In this section, we prove security for the following function f:

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

This function has an embedded XOR, and is defined over a finite domain so that $X_\kappa = X = \{x_1, x_2, x_3\}$ and $Y_\kappa = Y = \{y_1, y_2\}$. For this f, we set $\alpha = 1/5$ in Protocol Π_{EXOR}.

Theorem 3.2.1 *If* (Gen, Mac, Vrfy) *is an information-theoretically secure m-time MAC, and π securely computes* ShareGen *with abort, then the protocol in Figure 3.4, with $\alpha = 1/5$, securely com-*

putes f with complete fairness.

Proof: Let Π denote the protocol in Figure 3.4 with $\alpha = 1/5$. We analyze Π in a hybrid model where there is a trusted party computing ShareGen. (One again, we stress that since π is only guaranteed to securely compute ShareGen with abort, the adversary is allowed to abort the trusted party computing ShareGen before it sends output to the honest party.) We will prove that an execution of Protocol Π_{EXOR} in this hybrid model is *statistically close* to an evaluation of f in the ideal model with complete fairness, where the only differences can occur due to MAC forgeries. Applying Proposition 2.7.1 then implies the theorem.

In the two claims that follow, we separately analyze corruption of P_2 and P_1. The case of a corrupted P_2 is relatively easy to analyze since P_1 always "gets the output first" (because, in every iteration — and iteration i^* in particular — P_2 sends its share first). The proof of security when P_1 is corrupted is much more challenging, and is given second.

Claim 3.2.2 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_2 and running Π in a hybrid model with access to an ideal functionality computing* ShareGen *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_2 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x,y,\kappa)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*,\kappa\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}'}_{\Pi,\mathcal{A}(z)}(x,y,\kappa)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*,\kappa\in\mathbb{N}}.$$

Proof: Let P_2 be corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input y, the auxiliary input z, and the security parameter κ. The simulator also chooses $\hat{y} \in Y$ uniformly at random. (It will send $\hat{y}$ to the trusted party, if needed.)

2. $\mathcal{S}$ receives the input y' of $\mathcal{A}$ to the computation of the functionality ShareGen.

   (a) If $y' \notin Y$ (this includes the case when $y' = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen and sends $\hat{y}$ to the trusted party computing f. It then halts and outputs whatever $\mathcal{A}$ outputs.

   (b) Otherwise, if the input is some $y' \in Y$, then $\mathcal{S}$ chooses uniformly-distributed shares $a_1^{(2)}, \ldots, a_m^{(2)}$ and $b_1^{(2)}, \ldots, b_m^{(2)}$. In addition, it generates keys $k_a, k_b \leftarrow \mathsf{Gen}(1^\kappa)$ and computes $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ for every i. Finally, it hands $\mathcal{A}$ the strings $b_1^{(2)}, \ldots, b_m^{(2)}$, $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, and k_b as its output from the computation of ShareGen.

3. If $\mathcal{A}$ sends abort to the trusted party computing ShareGen, then $\mathcal{S}$ sends $\hat{y}$ to the trusted party computing f. It then halts and outputs whatever $\mathcal{A}$ outputs. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.

4. $\mathcal{S}$ chooses i^* according to a geometric distribution with parameter α.

5. For $i = 1$ to $i^* - 1$:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_i^{(2)}, \hat{t}_i^a)$ in the ith iteration. If $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends $\hat{y}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise, $\mathcal{S}$ proceeds.

(b) $\mathcal{S}$ chooses $\hat{x} \in X$ uniformly at random, computes $b_i = f(\hat{x}, y')$, sets $b_i^{(1)} = b_i^{(2)} \oplus b_i$, and computes $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$. It gives $\mathcal{A}$ the message $(b_i^{(1)}, t_i^b)$. (Note that a fresh $\hat{x}$ is chosen in every iteration.)

6. For $i = i^*$:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_{i^*}^{(2)}, \hat{t}_{i^*}^a)$. If $\mathsf{Vrfy}_{k_a}(i^* \| \hat{a}_{i^*}^{(2)}, \hat{t}_{i^*}^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends $\hat{y}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise, $\mathcal{S}$ sends y' to the trusted party computing f, receives the output $z = f(x, y')$, and proceeds.

   (b) $\mathcal{S}$ sets $b_{i^*}^{(1)} = b_{i^*}^{(2)} \oplus z$, and computes $t_{i^*}^b = \mathsf{Mac}_{k_b}(i^* \| b_{i^*}^{(1)})$. It gives $\mathcal{A}$ the message $(b_{i^*}^{(1)}, t_{i^*}^b)$.

7. For $i = i^* + 1$ to m:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_i^{(2)}, \hat{t}_i^a)$ in the ith iteration. If $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) $\mathcal{S}$ sets $b_i^{(1)} = b_i^{(2)} \oplus z$, and computes $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$. It gives $\mathcal{A}$ the message $(b_i^{(1)}, t_i^b)$.

8. If $\mathcal{S}$ has not halted yet, at this point it outputs whatever $\mathcal{A}$ outputs and halts.

We assume that if $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 1$, then $\hat{a}_i^{(2)} = a_i^{(2)}$ (meaning that $\mathcal{A}$ sent the same share that it received). It is straightforward to prove that this is the case with all but negligible probability based on the information-theoretic security of the MAC. Under this assumption, the distribution generated by $\mathcal{S}$ in an ideal-world execution with a trusted party computing f is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_1. To see this, first note that the view of $\mathcal{A}$ is identical in both worlds. As for the output of P_1, if $\mathcal{A}$ aborts (or sends an invalid message) before sending its first-iteration message, then P_1 outputs $f(x, \hat{y})$ for a random $\hat{y} \in Y$ in both the hybrid and ideal worlds. If $\mathcal{A}$ aborts after sending a valid iteration-i message then, conditioned on $\mathcal{A}$'s view at that point, the distribution of i^* is identical in the hybrid and ideal worlds. Moreover, in both worlds, P_1 outputs $f(x, \hat{y})$ (for a random $\hat{y} \in Y$) if $i < i^*$ and outputs $f(x, y')$ if $i \geq i^*$. This concludes the proof of this case. ■

We remark that the proof of the preceding claim did not depend on the value of α or the particular function f. The value of α and the specific nature of f will become important when we deal with a malicious P_1 in the proof of the following claim.

Claim 3.2.3 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_1 and running Π in a hybrid model with access to an ideal functionality computing* ShareGen *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x, y, \kappa)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*, \kappa\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}'}_{\Pi,\mathcal{A}(z)}(x, y, \kappa)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*, \kappa\in\mathbb{N}}.$$

Proof Sketch: The proof of the above claim is the most technical part of this section. We begin by giving some intuition before proceeding with the proof. Consider first the simulator from the proof of Claim 3.2.2 (adapted in the natural way for P_1). The simulation succeeds as long as the adversary aborts before or after round i^*. However, if the adversary aborts exactly in

round i^*, the simulation fails. As before, the simulator will have to send x' to the trusted party at the beginning of this round in order to learn the correct output value. Then, because the ideal world is fair, P_2 *also* receives correct output. In contrast, in the real world protocol, if P_1 aborts in round i^*, P_2 outputs b_{i^*-1}, which is a random output value, and is not necessarily correct.

The key to fixing this problem comes from the following crucial observation: the adversary does not know for certain that he aborted in round i^*! In particular, the exact same view that caused P_1 to abort in round i^* is nearly as likely to occur when i^* happens to be set to some larger value. Since P_1's view is identical when this occurs, his actions must also be identical, leading him to abort now in some round *prior* to i^*. When the adversary aborts prior to i^*, the simulator in the proof of Claim 3.2.2 submits a random value $\hat{x} \in X$ to the trusted party, resulting in random output for P_2, just as in the real world. However, in the following proof, we will *not* submit a uniformly chosen value on behalf of P_1 in this case. As we have just described, if P_1 aborts exactly in round i^*, our simulation fails because it yields correct output for P_2 too often. When P_1 aborts the protocol in some round prior to i^*, we will compensate for the above fact by choosing $\hat{x}$ from a distribution that yields correct output for P_2 with *smaller* probability than does the uniform distribution. By choosing $\hat{x}$ from just the right distribution, we can guarantee that the overall probability that P_2 outputs the correct value (conditioned on the view of P_1) is the same in both the ideal and real worlds. Of course, the probabilities are split differently when we condition on whether P_1 aborts in round i^*, but this fact is unrecognizable when observing only the transcript. The technical proof follows.

Proof: Say P_1 is corrupted by an adversary $\mathcal{A}$. We construct a simulator $\mathcal{S}$ that is given black-box access to $\mathcal{A}$. *For readability in what follows, we ignore the presence of the MAC-tags and keys.* That is, we do not mention the fact that $\mathcal{S}$ computes MAC-tags for messages it gives to $\mathcal{A}$, nor do we mention the fact that $\mathcal{S}$ must verify the MAC-tags on the messages sent by $\mathcal{A}$. When we say that $\mathcal{A}$ "aborts", we include in this the event that $\mathcal{A}$ sends an invalid message, or a message whose tag does not pass verification.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input[2] x', auxiliary input z, and the security parameter κ. The simulator also chooses $\hat{x} \in X$ uniformly at random (it will send $\hat{x}$ to the trusted party, if needed).

2. $\mathcal{S}$ receives the input x of $\mathcal{A}$ to the computation of the functionality ShareGen.

   (a) If $x \notin X$ (this includes the case when $x = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen, sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) Otherwise, if the input is some $x \in X$, then $\mathcal{S}$ chooses uniformly-distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. Then, $\mathcal{S}$ gives these shares to $\mathcal{A}$ as its output from the computation of ShareGen.

3. If $\mathcal{A}$ sends abort to the trusted party computing ShareGen, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.

4. Choose i^* according to a geometric distribution with parameter α. We now branch depending on the value of x.

[2] To simplify readability later, we reserve x for the value input by $\mathcal{A}$ to the computation of ShareGen.

If $x = x_3$:

5. For $i = 1$ to m:
   (a) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus 1$ and gives $a_i^{(2)}$ to $\mathcal{A}$. (Recall that $f(x_3, y) = 1$ for any y.)
   (b) If $\mathcal{A}$ aborts and $i \leq i^*$, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f. If $\mathcal{A}$ aborts and $i > i^*$ then $\mathcal{S}$ sends $x = x_3$ to the trusted party computing f. In either case, $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts.

   If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds to the next iteration.
6. If $\mathcal{S}$ has not halted yet, then if $i^* \leq m$ it sends x_3 to the trusted party computing f while if $i^* > m$ it sends $\hat{x}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

If $x \in \{x_1, x_2\}$:

7. Let $\bar{x}$ be the "other" value in $\{x_1, x_2\}$; i.e., if $x = x_c$ then $\bar{x} = x_{3-c}$.
8. For $i = 1$ to $i^* - 1$:
   (a) $\mathcal{S}$ chooses $\hat{y} \in Y$ uniformly at random, computes $a_i = f(x, \hat{y})$, and sets $a_i^{(2)} = a_i^{(1)} \oplus a_i$. It gives $a_i^{(2)}$ to $\mathcal{A}$. (Note that a fresh $\hat{y}$ is chosen in every iteration.)
   (b) If $\mathcal{A}$ aborts:
      i. If $a_i = 0$, then with probability $1/3$ send $\bar{x}$ to the trusted party computing f, and with probability $2/3$ send x_3.
      ii. If $a_i = 1$, then with probability $1/3$ send x to the trusted party computing f; with probability $1/2$ send $\bar{x}$; and with probability $1/6$ send x_3.

   In either case, $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts.

   If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.
9. For $i = i^*$ to m:
   (a) If $i = i^*$ then $\mathcal{S}$ sends x to the trusted party computing f and receives $z = f(x, y)$.
   (b) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus z$ and gives $a_i^{(2)}$ to $\mathcal{A}$.
   (c) If $\mathcal{A}$ aborts, then $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.
10. If $\mathcal{S}$ has not yet halted, and has not yet sent anything to the trusted party computing f (this can only happen if $i^* > m$ and $\mathcal{A}$ has never aborted), then it sends $\hat{x}$ to the trusted party. Then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

We will show that the distribution generated by $\mathcal{S}$ in an ideal-world execution with a trusted party computing f is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_2. (As always, we are ignoring here the possibility that $\mathcal{A}$ can forge a valid MAC-tag; once again, this introduces only a negligible statistical difference.) We first observe that the case of $x = x_3$ is straightforward since in this case $\mathcal{S}$ does not need to send anything to the trusted party until *after* $\mathcal{A}$ aborts. (This is because $a_i = 1$ for all i since $f(x_3, y) = 1$ for all $y \in Y$; note that this is the first time in the proof we rely on specific properties of f.) For the remainder of the proof, we therefore focus our attention on the case when $x \in \{x_1, x_2\}$.

Let $\text{VIEW}_{\mathsf{hyb}}(x, y)$ be the random variable denoting the view of $\mathcal{A}$ in the hybrid world (i.e., running Π with a trusted party computing ShareGen) when P_2 holds input y and $\mathcal{A}$ uses input x in the computation of ShareGen. Let $\text{VIEW}_{\mathsf{ideal}}(x, y)$ be the random variable denoting the view of $\mathcal{A}$ in the ideal world (i.e., where $\mathcal{S}$ runs $\mathcal{A}$ as a black-box and interacts with a trusted party computing f) with x, y similarly defined. Finally, let $\text{OUT}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)$ be random variables denoting the output of the honest player P_2 in the hybrid and ideal worlds, respectively, for the given x and y. We will show that for any $x \in \{x_1, x_2\}$ and $y \in Y$,

$$\left(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{hyb}}(x, y)\right) \equiv \left(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)\right). \tag{3.2}$$

(We stress that the above assumes $\mathcal{A}$ never forges a valid MAC-tag, and therefore the security parameter κ can be ignored and perfect equivalence obtained. Taking into account this possibility, the above distributions would then have statistical difference negligible in the security parameter κ.) It is immediate from the description of $\mathcal{S}$ that $\text{VIEW}_{\mathsf{hyb}}(x, y) \equiv \text{VIEW}_{\mathsf{ideal}}(x, y)$ for any x, y; the difficulty lies in arguing about the joint distribution of $\mathcal{A}$'s view and P_2's output, as above.

We prove Eq. (3.2) by showing that for any x, y as above and any view v and bit b, it holds that:

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{hyb}}(x, y)\right) = (v, b)\right] \\ &= \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)\right) = (v, b)\right]. \end{aligned} \tag{3.3}$$

Clearly, if v represents a view that does not correspond to the actions of $\mathcal{A}$ (e.g., v contains a_i, but given view v the adversary would have aborted prior to iteration i; or v does not contain a_i, but given view v the adversary would not have aborted prior to iteration i), then both probabilities in Eq. (3.3) are identically 0 (regardless of b). Going forward, we therefore only consider views that correspond to actions of $\mathcal{A}$.

$\mathcal{A}$'s view consists of its initial inputs, the values $a_1^{(1)}, b_1^{(1)}, \ldots, a_m^{(1)}, b_m^{(1)}$ that $\mathcal{A}$ receives from computation of ShareGen, and — if $\mathcal{A}$ does not abort before the first iteration — a sequence of values $a_1, \ldots, a_i$ where i is the iteration in which $\mathcal{A}$ aborts (if any). (Technically $\mathcal{A}$ receives $a_1^{(2)}, \ldots, a_i^{(2)}$ but we equivalently consider the reconstructed values $a_1, \ldots, a_i$ instead.) Looking at the description of $\mathcal{S}$, it is easy to see that if v represents a view in which $\mathcal{A}$ aborts before the first iteration, or in which $\mathcal{A}$ never aborts (i.e., $\mathcal{A}$ runs the protocol to completion), then Eq. (3.3) holds for either choice of b. Thus, the "difficult" cases to analyze are exactly those in which $\mathcal{A}$ aborts in some iteration i.

Let v be a view in which $\mathcal{A}$ aborts in iteration i (i.e., after receiving its iteration-i message). We will let $\mathcal{A}$'s initial inputs and its outputs from ShareGen be implicit, and focus on the vector of values $\vec{a}_i = (a_1, \ldots, a_i)$ that $\mathcal{A}$ sees before it aborts in iteration i, We will show that for any x, y as above, any $\vec{a}_i$, and any bit b it holds that

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{hyb}}(x, y)\right) = (\vec{a}_i, b)\right] \\ &= \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)\right) = (\vec{a}_i, b)\right]. \end{aligned} \tag{3.4}$$

We stress that we are considering exactly those views $\vec{a}_i = (a_1, \ldots, a_i)$ in which $\mathcal{A}$ aborts after receiving its iteration-i message; there is thus no possibility that $\mathcal{A}$ might abort given the sequence of values $a_1, \ldots, a_j$ (with $j < i$).

Toward proving Eq. (3.4), we first prove:

Claim 3.2.4 *For any $x \in \{x_1, x_2\}$ and $y \in Y$,*

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* < i\right] \\ &\quad = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* < i\right]. \end{aligned} \tag{3.5}$$

Proof: A proof of this claim follows easily from the observation that, conditioned on $i^* < i$, the "true" input of P_1 is used to compute P_2's output in both the hybrid and ideal worlds.

Formally, fix some x, y and let these be implicit in what follows. To prove the claim, note that

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (\vec{a}_i, b) \bigwedge i^* < i\right] \\ &\quad = \Pr\left[\text{OUT}_{\mathsf{hyb}} = b \mid \text{VIEW}_{\mathsf{hyb}} = \vec{a}_i \bigwedge i^* < i\right] \cdot \Pr\left[\text{VIEW}_{\mathsf{hyb}} = \vec{a}_i \bigwedge i^* < i\right]\end{aligned}$$

and

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (\vec{a}_i, b) \bigwedge i^* < i\right] \\ &\quad = \Pr\left[\text{OUT}_{\mathsf{ideal}} = b \mid \text{VIEW}_{\mathsf{ideal}} = \vec{a}_i \bigwedge i^* < i\right] \cdot \Pr\left[\text{VIEW}_{\mathsf{ideal}} = \vec{a}_i \bigwedge i^* < i\right].\end{aligned}$$

It follows from the description of $\mathcal{S}$ that

$$\Pr\left[\text{VIEW}_{\mathsf{hyb}} = \vec{a}_i \bigwedge i^* < i\right] = \Pr\left[\text{VIEW}_{\mathsf{ideal}} = \vec{a}_i \bigwedge i^* < i\right].$$

Furthermore, conditioned on $i^* < i$ the output of P_2 is the correct output $f(x,y)$ in both the hybrid and ideal worlds. We conclude that Eq. (3.5) holds. ■

To complete the proof of Eq. (3.4), we prove that for any $x \in \{x_1, x_2\}$ and $y \in Y$, any $\vec{a}_i \in \{0,1\}^i$, and all $b \in \{0,1\}$ it holds that

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right] \\ &\quad = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right].\end{aligned} \tag{3.6}$$

This is the crux of the proof. Write $\vec{a}_i = (\vec{a}_{i-1}, a)$, $\text{VIEW}_{\mathsf{hyb}} = (\overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}}, \text{VIEW}^i_{\mathsf{hyb}})$, and $\text{VIEW}_{\mathsf{ideal}} = (\overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}}, \text{VIEW}^i_{\mathsf{ideal}})$. (In what follows, we also often leave x and y implicit in the interests of readability.) Then

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right] \\ &\quad = \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (a, b) \mid \overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right] \cdot \Pr\left[\overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right]\end{aligned}$$

and

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right] \\ &\quad = \Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (a, b) \mid \overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right] \cdot \Pr\left[\overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right].\end{aligned}$$

Once again, it follows readily from the description of $\mathcal{S}$ that

$$\Pr\left[\overline{\text{VIEW}}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1} \bigwedge i^* \geq i\right] = \Pr\left[\overline{\text{VIEW}}_{\mathsf{ideal}}^{i-1} = \vec{a}_{i-1} \bigwedge i^* \geq i\right].$$

Moreover, conditioned on the event that $i^* \geq i$, the random variables of $\text{VIEW}_{\mathsf{hyb}}^i$ and $\text{OUT}_{\mathsf{hyb}}$ (resp., $\text{VIEW}_{\mathsf{ideal}}^i$ and $\text{OUT}_{\mathsf{ideal}}$) are independent of $\overline{\text{VIEW}}_{\mathsf{hyb}}^{i-1}$ (resp., $\overline{\text{VIEW}}_{\mathsf{ideal}}^{i-1}$) for fixed x and y. Thus, Eq. (3.6) is proved once we show that

$$\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}^i, \text{OUT}_{\mathsf{hyb}}\right) = (a,b) \mid i^* \geq i\right] = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}^i, \text{OUT}_{\mathsf{ideal}}\right) = (a,b) \mid i^* \geq i\right] \quad (3.7)$$

for all x, y, a, b as above. We prove this via case-by-case analysis. For convenience, we recall the table for f:

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

Case 1: $x = x_1$ and $y = y_1$. We analyze the hybrid world first, followed by the ideal world.

Hybrid world. We first consider the hybrid world where the parties are running protocol Π. If $\mathcal{A}$ aborts after receiving its iteration-i message, P_2 will output $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$. Since $i^* \geq i$, we have $b_{i-1} = f(\hat{x}, y_1)$ where $\hat{x}$ is chosen uniformly from X. So $\Pr[\text{OUT}_{\mathsf{hyb}} = 0] = \Pr_{\hat{x} \leftarrow X}[f(\hat{x}, y_1) = 0] = 1/3$ and $\Pr[\text{OUT}_{\mathsf{hyb}} = 1] = 2/3$.

Since $i^* \geq i$, the value of $\text{VIEW}_{\mathsf{hyb}}^i = a_i$ is independent of the value of b_{i-1}. Conditioned on the event that $i^* \geq i$, we have $\Pr[i^* = i] = \alpha = 1/5$ and $\Pr[i^* > i] = 4/5$. If $i^* = i$, then $a_i = f(x,y) = f(x_1, y_1) = 0$. If $i^* > i$, then $a_i = f(x_1, \hat{y})$ where $\hat{y}$ is chosen uniformly from Y. So $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. Overall, then, we have

$$\begin{aligned}
\Pr[\text{VIEW}_{\mathsf{hyb}}^i(x_1, y_1) = 0 \mid i^* \geq i] &= \alpha \cdot 1 + (1-\alpha) \cdot \frac{1}{2} = \frac{3}{5} \\
\Pr[\text{VIEW}_{\mathsf{hyb}}^i(x_1, y_1) = 1 \mid i^* \geq i] &= \alpha \cdot 0 + (1-\alpha) \cdot \frac{1}{2} = \frac{2}{5}.
\end{aligned}$$

Putting everything together gives

$$\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}^i(x_1, y_1), \text{OUT}_{\mathsf{hyb}}(x_1, y_1)\right) = (a,b) \mid i^* \geq i\right] = \begin{cases} \frac{3}{5} \cdot \frac{1}{3} = \frac{1}{5} & (a,b) = (0,0) \\ \frac{3}{5} \cdot \frac{2}{3} = \frac{2}{5} & (a,b) = (0,1) \\ \frac{2}{5} \cdot \frac{1}{3} = \frac{2}{15} & (a,b) = (1,0) \\ \frac{2}{5} \cdot \frac{2}{3} = \frac{4}{15} & (a,b) = (1,1) \end{cases} \quad (3.8)$$

Ideal world. We now turn our attention to the ideal world. Since we are conditioning on $i^* \geq i$, here it is also the case that $\Pr[i^* = i] = \alpha = 1/5$ and $\Pr[i^* > i] = 4/5$. Furthermore, if $i^* = i$ then $\text{VIEW}_{\mathsf{ideal}}^i = a_i = f(x_1, y_1) = 0$. Now, however, if $i^* = i$ then $\mathcal{S}$ has already sent x_1 to the trusted party computing f (in order to learn the value $f(x_1, y_1)$) and so P_2 will *also* output $f(x_1, y_1) = 0$, rather than some independent value b_{i-1}.

When $i^* > i$, then (by construction of $\mathcal{S}$) we have $\Pr[a_i = 0] = \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 0] = 1/2$ and $\Pr[a_i = 1] = 1/2$. Now, however, the output of P_2 depends on the value sent to the trusted party following an abort by $\mathcal{A}$, which in turn depends on a_i (cf. step 8(b) of $\mathcal{S}$). In

particular, we have:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1,y_1)=0 \mid a_i=0 \bigwedge i^*>i]\\ &\quad = \quad \Pr[\mathcal{S} \text{ sends } x_1 \text{ to the trusted party} \mid a_i=0 \bigwedge i^*>i] \quad = \quad 0,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1,y_1)=0 \mid a_i=1 \bigwedge i^*>i]\\ &\quad = \quad \Pr[\mathcal{S} \text{ sends } x_1 \text{ to the trusted party} \mid a_i=1 \bigwedge i^*>i] \quad = \quad 1/3\end{aligned}$$

(in calculating the above, recall that $x = x_1$). Putting everything together, we obtain

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,0) \mid i^*\geq i\right]\\ &= \quad \alpha\cdot\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,0) \mid i^*=i\right]\\ &\qquad +(1-\alpha)\cdot\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,0) \mid i^*>i\right]\\ &= \quad \alpha+(1-\alpha)\cdot 0 \quad = \quad \frac{1}{5}\,.\end{aligned} \tag{3.9}$$

Similarly,

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,1) \mid i^*\geq i\right] \quad = \quad (1-\alpha)\cdot\frac{1}{2}\cdot 1 \quad = \quad \frac{2}{5} \tag{3.10}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(1,0) \mid i^*\geq i\right] \quad = \quad (1-\alpha)\cdot\frac{1}{2}\cdot\frac{1}{3} \quad = \quad \frac{2}{15} \tag{3.11}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(1,1) \mid i^*\geq i\right] \quad = \quad (1-\alpha)\cdot\frac{1}{2}\cdot\frac{2}{3} \quad = \quad \frac{4}{15}\,, \tag{3.12}$$

in exact agreement with Eq. (3.8).

Case 2: $x = x_2$ and $y = y_1$. In all the remaining cases, the arguments are the same as before; just the numbers differ. Therefore, we will allow ourselves to be more laconic.

In the hybrid world, conditioned on $i^* \geq i$, the values of $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$ and $\text{VIEW}^i_{\mathsf{hyb}} = a_i$ are again independent. The distribution of b_{i-1} is given by: $\Pr[b_{i-1}=0] = \Pr_{\hat{x}\leftarrow X}[f(\hat{x},y_1)=0] = 1/3$ and $\Pr[b_{i-1}=1] = 2/3$. As for the distribution of a_i, we have

$$\begin{aligned}\Pr[a_i=1 \mid i^*\geq i] \quad &= \quad \alpha\cdot\Pr[a_i=1 \mid i^*=i]+(1-\alpha)\cdot\Pr[a_i=1 \mid i^*>i]\\ &= \quad \alpha\cdot 1+(1-\alpha)\cdot\Pr_{\hat{y}\leftarrow Y}[f(x_1,\hat{y})=1]\\ &= \quad \frac{1}{5}+\frac{4}{5}\cdot\frac{1}{2} \quad = \quad \frac{3}{5},\end{aligned}$$

and so $\Pr[a_i = 0 \mid i^* \geq i] = 2/5$. Putting everything together gives

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x_2,y_1),\text{OUT}_{\mathsf{hyb}}(x_2,y_1)\right)=(a,b) \mid i^*\geq i\right] = \begin{cases}\frac{2}{5}\cdot\frac{1}{3}=\frac{2}{15} & (a,b)=(0,0)\\ \frac{2}{5}\cdot\frac{2}{3}=\frac{4}{15} & (a,b)=(0,1)\\ \frac{3}{5}\cdot\frac{1}{3}=\frac{1}{5} & (a,b)=(1,0)\\ \frac{3}{5}\cdot\frac{2}{3}=\frac{2}{5} & (a,b)=(1,1)\end{cases} \tag{3.13}$$

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x_2, y_1) = 1$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. The value of $\text{OUT}_{\mathsf{ideal}}$ is now dependent on the value of a_i (cf. step 8(b) of $\mathcal{S}$); specifically:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_1) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_1 \text{ to the trusted party} \mid a_i = 0 \bigwedge i^* > i] \;=\; 1/3,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_1) = 0 \mid a_i = 1 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_1 \text{ to the trusted party} \mid a_i = 1 \bigwedge i^* > i] \;=\; 1/2\end{aligned}$$

(using the fact that $x = x_2$). Putting everything together, we obtain

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (0,0) \mid i^* \geq i\right] = (1-\alpha) \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{2}{15} \tag{3.14}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (0,1) \mid i^* \geq i\right] = (1-\alpha) \cdot \frac{1}{2} \cdot \frac{2}{3} = \frac{4}{15} \tag{3.15}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (1,0) \mid i^* \geq i\right] = (1-\alpha) \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{5} \tag{3.16}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (1,1) \mid i^* \geq i\right] = \alpha + (1-\alpha) \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{2}{5}, \tag{3.17}$$

in exact agreement with Eq. (3.13).

Case 3: $x = x_1$ and $y = y_2$. In the hybrid world, this case is exactly symmetric to the case when $x = x_2$ and $y = y_1$. Thus we obtain the same distribution as in Eq. (3.13).

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x_1, y_2) = 1$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_2, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. The value of $\text{OUT}_{\mathsf{ideal}}$ is dependent on the value of a_i (cf. step 8(b) of $\mathcal{S}$); specifically:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1, y_2) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 0 \bigwedge i^* > i] \;=\; 1/3,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1, y_2) = 0 \mid a_i = 1 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 1 \bigwedge i^* > i] \;=\; 1/2\end{aligned}$$

(using the fact that $x = x_1$). Putting everything together, we obtain the same distribution as in Eqs. (3.14)–(3.17). The distributions in the hybrid and ideal worlds are, once again, in exact agreement.

Case 4: $x = x_2$ and $y = y_2$. In the hybrid world, this case is exactly symmetric to the case when $x = x_1$ and $y = y_1$. Thus we obtain the same distribution as in Eq. (3.8).

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x_2, y_2) = 0$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_2, \hat{y}) = 1] = 1/2$ and

$\Pr[a_i = 0] = 1/2$. The value of $\text{OUT}_{\mathsf{ideal}}$ is dependent on the value of a_i (cf. step 8(b) of $\mathcal{S}$); specifically:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_2) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 0 \bigwedge i^* > i] \;=\; 0,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_2) = 0 \mid a_i = 1 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 1 \bigwedge i^* > i] \;=\; 1/3\end{aligned}$$

(using the fact that $x = x_2$). Putting everything together, we obtain the same distribution as in Eqs. (3.9)–(3.12). The distributions in the hybrid and ideal worlds are, once again, in exact agreement. This completes the proof of Claim 3.2.3. ■

The preceding claims along with Proposition 2.7.1 conclude the proof of Theorem 3.2.1. ■

3.2.2 Extending the Protocol to Other Functions

Before specifying the more general functions for which Protocol Π_{EXOR} (cf. Figure 3.4) can be applied, we briefly discuss how we chose the value $\alpha = 1/5$ for the specific f of Section 3.2.1. This will provide some intuition that will be helpful in the section that follows. It should be clear that our entire discussion in this subsection assumes the specific simulation strategy described in the proof of Theorem 3.2.1. It may be the case that a different simulation strategy would allow for other values of α, or there may exist a different protocol altogether for computing f.

Consider the case of a malicious P_1 who aborts after receiving its iteration-i message, and let the parties' inputs be $x = x_1$, $y = y_1$ (note $f(x_1, y_1) = 0$). We use the notation as in the proof of Claim 3.2.3, so that $\text{VIEW}^i_{\mathsf{hyb}}$ denotes the value a_i that P_1 reconstructs in iteration i and $\text{OUT}_{\mathsf{hyb}}$ denote the output of the honest P_2. The protocol itself ensures that in the hybrid world we have

$$\begin{aligned}&\Pr[\big(\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1), \text{OUT}_{\mathsf{hyb}}(x_1, y_1)\big) = (0, 0) \mid i^* \geq i] \\ &\quad = \Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* \geq i] \cdot \Pr[\text{OUT}_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* \geq i],\end{aligned}$$

since $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$ is independent of $\text{VIEW}^i_{\mathsf{hyb}} = a_i$ when $i^* \geq i$. We have

$$\Pr[\text{OUT}_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* \geq i] = \Pr_{\hat{x} \leftarrow X}[f(\hat{x}, y_1) = 0] = 1/3$$

and

$$\begin{aligned}&\Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* \geq i] \\ &\quad = \alpha \cdot \Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* = i] + (1 - \alpha) \cdot \Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* > i] \\ &\quad = \alpha + (1 - \alpha) \cdot \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 0] \\ &\quad = \alpha + (1 - \alpha) \cdot \frac{1}{2},\end{aligned}$$

where the first equality holds since $\Pr[i^* = i \mid i^* \geq i] = \alpha$. Putting everything together we see that

$$\Pr[\big(\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1), \text{OUT}_{\mathsf{hyb}}(x_1, y_1)\big) = (0,0) \mid i^* \geq i] = \frac{1}{3} \cdot \left(\alpha + (1-\alpha) \cdot \frac{1}{2}\right).$$

In the ideal world, our simulation strategy ensures that, conditioned on $i^* \geq i$, the simulator $\mathcal{S}$ sends $x = x_1$ to the trusted party with probability α; when this occurs, the simulator will then set $\text{VIEW}^i_{\mathsf{ideal}} = a_i = f(x_1, y_1) = 0$, and the honest party P_2 will output $f(x_1, y_1) = 0$. Therefore, regardless of anything else the simulator might do,

$$\Pr[\big(\text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1), \text{OUT}_{\mathsf{ideal}}(x_1, y_1)\big) = (0,0) \mid i^* \geq i] \;\geq\; \alpha.$$

If we want the ideal-world and hybrid-world distributions to be equal, then this requires

$$\alpha \leq \left(\alpha + (1-\alpha) \cdot \frac{1}{2}\right) \cdot \frac{1}{3},$$

which is equivalent to requiring $\alpha \leq 1/5$. A similar argument applied to the other possible values for x, y shows that $\alpha \leq 1/5$ suffices for all of them. Setting $\alpha = 1/5$ minimizes the number of rounds of the protocol.

Having fixed the value of α, we now explain how we determined the simulator's actions (for a malicious P_1) in step 8(b). We begin by introducing some notation that we will also use in the following section.

Define $p_{x_i} \stackrel{\text{def}}{=} \Pr_{\hat{y} \leftarrow Y}[f(x_i, \hat{y}) = 1]$ and, similarly, define $p_{y_i} \stackrel{\text{def}}{=} \Pr_{\hat{x} \leftarrow X}[f(\hat{x}, y_i) = 1]$. Let x' be as in the description of $\mathcal{S}$ in the proof of Claim 3.2.3. If $\mathcal{A}$ aborts in round $i < i^*$ after receiving the bit a_i, then we denote the event that $\mathcal{S}$ sends x_i to the ideal functionality computing f by $X^{(a_i)}_{x' \to x_i}$. Using this notation, we have from step 8(b) of $\mathcal{S}$ that:

$$\Pr[X^{(1)}_{x_1 \to x_1}] = \frac{1}{3} \qquad \Pr[X^{(1)}_{x_1 \to x_2}] = \frac{1}{2} \qquad \Pr[X^{(1)}_{x_1 \to x_3}] = \frac{1}{6}.$$

Consider once again the case $x = x_1$ and $y = y_1$. In the hybrid world, by construction of Protocol Π_{EXOR}, we have

$$\begin{aligned}
&\Pr[\big(\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1), \text{OUT}_{\mathsf{hyb}}(x_1, y_1)\big) = (1,1) \mid i^* \geq i] \\
&\quad = \Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 1 \mid i^* \geq i] \cdot \Pr[\text{OUT}_{\mathsf{hyb}}(x_1, y_1) = 1 \mid i^* \geq i] \\
&\quad = (1-\alpha) \cdot p_{x_1} \cdot p_{y_1}.
\end{aligned}$$

(Note that if $i^* = i$, which occurs with probability α, then $a_i = f(x_1, y_1) = 0$.) Because of the way $\mathcal{S}$ is defined, in the ideal world we have

$$\begin{aligned}
&\Pr[\big(\text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1), \text{OUT}_{\mathsf{ideal}}(x_1, y_1)\big) = (1,1) \mid i^* \geq i] \\
&\quad = \Pr[\text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1) = 1 \mid i^* \geq i] \cdot \Pr[\text{OUT}_{\mathsf{ideal}}(x_1, y_1) = 1 \mid \text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1) = 1 \bigwedge i^* \geq i] \\
&\quad = (1-\alpha) \cdot p_{x_1} \cdot \Big(\Pr[X^{(1)}_{x_1 \to x_2}] + \Pr[X^{(1)}_{x_1 \to x_3}]\Big).
\end{aligned}$$

If we want these to be equal, this requires $\Pr[X^{(1)}_{x_1 \to x_2}] + \Pr[X^{(1)}_{x_1 \to x_3}] = p_{y_1} = \frac{2}{3}$.

Proceeding similarly for the case when $x = x_1$ and $y = y_2$ and looking at the probability

that $a_i = 0$ and the output of P_2 is 1, we derive

$$\Pr[X^{(1)}_{x_1 \to x_1}] + \Pr[X^{(1)}_{x_1 \to x_3}] = \frac{\alpha \cdot (p_{y_2} - 1)}{(1-\alpha)(1-p_{x_1})} + p_{y_2} = \frac{1}{2}.$$

Combining the above with the constraint that $\Pr[X^{(1)}_{x_1 \to x_1}] + \Pr[X^{(1)}_{x_1 \to x_2}] + \Pr[X^{(1)}_{x_1 \to x_3}] = 1$ we obtain the unique feasible values used in step 8(b) of $\mathcal{S}$ (for the case $x = x_1$). The case of $x = x_2$ follows via a similar analysis.

Looking at the problem more generally, we observe that for certain functions f (e.g., the boolean XOR function), the problem is over-constrained and *no* feasible solution exists (regardless of the choice of α). In the following section we will argue that our protocol can be applied to any function f for which the above constraints can be satisfied for all possible inputs x, y.

3.2.3 A Characterization of When Protocol Π_{EXOR} is Secure

In this section we characterize a class of functions that can be securely computed with complete fairness using Protocol Π_{EXOR}. The proof is a generalization of the proof from Section 3.2.1.

Notation. We assume a single-output, boolean function $f : X \times Y \to \{0, 1\}$ defined over a finite domain, where $X = \{x_1, \ldots x_\ell\}$ and $Y = \{y_1, \ldots, y_m\}$. We let M_f denote the $\ell \times m$ matrix whose entry at position (i, j) is $f(x_i, y_j)$, and let $\mathbf{v}_y$ denote the column of M_f corresponding to the input y of P_2. For every input $x \in X$ of player P_1 we define

$$p_x \stackrel{\text{def}}{=} \Pr_{\hat{y} \leftarrow Y}[f(x, \hat{y}) = 1],$$

where $\hat{y}$ is chosen uniformly from the domain Y of player P_2. Equivalently, $p_x = \frac{\sum_{y \in Y} f(x,y)}{m}$. We define p_y, for $y \in Y$, symmetrically. In addition, let $\bar{p}_x \stackrel{\text{def}}{=} 1 - p_x$ and $\bar{p}_y \stackrel{\text{def}}{=} 1 - p_y$.

We set α as follows:

$$\alpha \stackrel{\text{def}}{=} \min_{(i,j)} \left\{ \frac{\left|1 - f(x_i, y_j) - p_{x_i}\right| \cdot \left|1 - f(x_i, y_j) - p_{y_j}\right|}{\left|1 - f(x_i, y_j) - p_{x_i}\right| \cdot \left|1 - f(x_i, y_j) - p_{y_j}\right| + \left|f(x_i, y_j) - p_{y_j}\right|} \right\}, \tag{3.18}$$

where the minimum is taken over $1 \leq i \leq \ell$ and $1 \leq j \leq m$. By simple calculation, one can show that $0 < \alpha \leq 1$ and, in fact, $\alpha < 1$ unless f is a constant function (in which case completely-fair computation of f is trivial). Using this value of α we define, for $x \in X$, the m-dimensional row vector $\vec{C}^{(0)}_x$, indexed by $y \in Y$, as follows:

$$\vec{C}^{(0)}_x(y) \stackrel{\text{def}}{=} \begin{cases} p_y & \text{if } f(x, y) = 1 \\ \frac{\alpha \cdot p_y}{(1-\alpha) \cdot \bar{p}_x} + p_y & \text{if } f(x, y) = 0 \end{cases}.$$

Similarly, we define $\vec{C}^{(1)}_x$ via:

$$\vec{C}^{(1)}_x(y) \stackrel{\text{def}}{=} \begin{cases} \frac{\alpha \cdot (p_y - 1)}{(1-\alpha) \cdot p_x} + p_y & \text{if } f(x, y) = 1 \\ p_y & \text{if } f(x, y) = 0 \end{cases}$$

(The denominators, above, are never 0.)

A row vector $(p_1, \ldots, p_\ell)$ of real numbers is a *probability vector* if $0 \leq p_i \leq 1$ for all i, and $\sum_i p_i = 1$. We are now ready to prove the following:

Theorem 3.2.5 *Let f be a single-output, boolean function, and let M_f and $\vec{C}_{x_i}^{(b)}$ be as defined above. If for all $b \in \{0,1\}$ and $x \in X$ there exists a probability vector $\vec{X}_x^{(b)} = (p_1, \ldots, p_\ell)$ such that*

$$\vec{X}_x^{(b)} \cdot M_f = \vec{C}_x^{(b)},$$

then there exists a protocol that securely computes f with complete fairness.

Proof: We take Protocol Π_{EXOR} with α computed as in Eq. (3.18). Simulation for a corrupted P_2 follows exactly along the lines of the proof of Claim 3.2.2; recall that the simulator in that case did not rely on any specific properties of the function f or the value of α. We therefore focus our attention on the case when the adversary $\mathcal{A}$ corrupts P_1. In this case, our simulator $\mathcal{S}$ is almost identical to the simulator described in the proof of Claim 3.2.3 (except, of course, that it uses the appropriate value of α); the only significant change is how we deal with an abort in iteration $i < i^*$ (this corresponds to step 8(b) in the simulator from the proof of Claim 3.2.3). For completeness, we describe the modified simulator in its entirety, although we once again ignore the presence of the MAC-tags and keys for simplicity.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input x', the auxiliary input, and the security parameter n. The simulator also chooses $\hat{x} \in X$ uniformly at random.

2. $\mathcal{S}$ receives the input x of $\mathcal{A}$ to the computation of the functionality ShareGen$'$.

   (a) If $x \notin X$, then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen$'$, sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) Otherwise, if the input is some $x \in X$, then $\mathcal{S}$ chooses uniformly-distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. Then, $\mathcal{S}$ gives these shares to $\mathcal{A}$ as its output from the computation of ShareGen$'$.

3. If $\mathcal{A}$ sends abort to the trusted party computing ShareGen$'$, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.

4. Choose i^* according to a geometric distribution with parameter α.

5. For $i = 1$ to $i^* - 1$:

   (a) $\mathcal{S}$ chooses $\hat{y} \in Y$ uniformly at random, computes $a_i = f(x, \hat{y})$, and sets $a_i^{(2)} = a_i^{(1)} \oplus a_i$. It gives $a_i^{(2)}$ to $\mathcal{A}$.

   (b) If $\mathcal{A}$ aborts, then $\mathcal{S}$ chooses x' according to the distribution defined by[3] $\vec{X}_x^{(a_i)}$, and sends x' to the trusted party computing f. It then outputs whatever $\mathcal{A}$ outputs, and halts.

   If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

6. For $i = i^*$ to m:

[3]This is understood in the natural way; i.e., x_j is chosen with probability $\vec{X}_x^{(a_i)}(j)$.

(a) If $i = i^*$ then $\mathcal{S}$ sends x to the trusted party computing f and receives $z = f(x, y)$.

(b) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus z$ and gives $a_i^{(2)}$ to $\mathcal{A}$.

(c) If $\mathcal{A}$ aborts, then $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

7. If $\mathcal{S}$ has not yet halted, and has not yet sent anything to the trusted party computing f (this can only happen if $i^* > m$ and $\mathcal{A}$ has not aborted), then it sends $\hat{x}$ to the trusted party. Then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

(The simulator constructed in Claim 3.2.3 branched depending on the value of x, but this was only a simplification due to the fact that the input x_3, there, completely determined the output. In general there need not be any such input.)

We borrow the same notation as in our proof of Claim 3.2.3. Examining that proof, we see that the proof here will proceed identically up to the point where we need to show that, for all inputs x, y and all $a, b \in \{0, 1\}$:

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (a, b) \mid i^* \geq i\right] = \Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (a, b) \mid i^* \geq i\right] \quad (3.19)$$

(This is Eq. (3.7) there. As was done there, we suppress explicit mention of the inputs when the notation becomes cumbersome.) We now fix arbitrary x, y and show that the above holds. We consider two sub-cases depending on the value of $f(x, y)$.

Case 1: x and y are such that $f(x, y) = 0$. In the hybrid world, when $\mathcal{A}$ aborts after receiving its iteration-i message, then P_2 outputs $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$ and the value of $\text{VIEW}^i_{\mathsf{hyb}} = a_i$ is independent of the value of b_{i-1}. By definition of the protocol, we have

$$\Pr[b_{i-1} = 0 \mid i^* \geq i] = \bar{p}_y \quad \text{and} \quad \Pr[b_{i-1} = 1 \mid i^* \geq i] = p_y,$$

since $b_{i-1} = f(\hat{x}, y)$ for $\hat{x}$ chosen uniformly from X. As for a_i, we have

$$\Pr\left[a_i = 0 \mid i^* \geq i\right] = \alpha + (1 - \alpha) \cdot \bar{p}_x \quad \text{and} \quad \Pr\left[a_i = 1 \mid i^* \geq i\right] = (1 - \alpha) \cdot p_x\,.$$

Since b_{i-1} and a_i are independent, we conclude that

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{hyb}}(x, y)\right) = (a, b) \mid i^* \geq i\right] = \begin{cases} (\alpha + (1 - \alpha) \cdot \bar{p}_x) \cdot \bar{p}_y & (a, b) = (0, 0) \\ (\alpha + (1 - \alpha) \cdot \bar{p}_x) \cdot p_y & (a, b) = (0, 1) \\ (1 - \alpha) \cdot p_x \cdot \bar{p}_y & (a, b) = (1, 0) \\ (1 - \alpha) \cdot p_x \cdot p_y & (a, b) = (1, 1) \end{cases}$$

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x, y) = 0$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 0] = \bar{p}_x$. The value of $\text{OUT}_{\mathsf{ideal}}$ is now dependent on the value of a_i (cf. step 5(b) of the simulator described in this section); specifically, we have:

$$\begin{aligned} &\Pr[\text{OUT}_{\mathsf{ideal}}(x, y) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &= \Pr[\mathcal{S} \text{ sends } x' \text{ to the trusted party s.t. } f(x', y) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &= \sum_{\bar{x}:\, f(\bar{x}, y) = 0} \Pr_{x' \leftarrow \vec{X}_x^{(0)}}[x' = \bar{x}] \end{aligned}$$

and, in the general case,

$$\Pr[\text{OUT}_{\mathsf{ideal}}(x,y) = b \mid a_i = a \bigwedge i^* > i] \;=\; \sum_{\bar{x}:\; f(\bar{x},y)=b} \Pr_{x' \leftarrow \vec{X}_x^{(a)}}[x' = \bar{x}].$$

We therefore have, for example,

$$\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,0) \mid i^* \geq i\right] \\
&\quad= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \sum_{\bar{x}:\; f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}_x^{(0)}}[x' = \bar{x}] \\
&\quad= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \left(1 - \vec{X}_x^{(0)} \cdot \mathbf{v}_y\right) \\
&\quad= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \left(1 - \vec{C}_x^{(0)}(y)\right) \\
&\quad= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \left(1 - \frac{\alpha\cdot p_y}{(1-\alpha)\cdot\bar{p}_x} - p_y\right) \\
&\quad= (\alpha + (1-\alpha)\cdot\bar{p}_x)\cdot\bar{p}_y, \;=\; \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (0,0) \mid i^* \geq i\right].
\end{aligned}$$

(The second equality uses the definitions of $\vec{X}_x^{(0)}$ and $\mathbf{v}_y$; the third equality uses the assumption, from the theorem, that $\vec{X}_x^{(0)} \cdot \mathbf{v}_y = \vec{C}_x^{(0)}(y)$. We then use the definition of $\vec{C}_x^{(0)}(y)$ and re-arrange using algebra.) This is equal to the associated probability in the hybrid world, as computed above.

For completeness, we include the calculations for the remaining cases:

$$\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,1) \mid i^* \geq i\right] \\
&\quad= (1-\alpha)\cdot\bar{p}_x \cdot \sum_{\bar{x}:\; f(\bar{x},y)=1} \Pr_{x' \leftarrow \vec{X}_x^{(0)}}[x' = \bar{x}] \\
&\quad= (1-\alpha)\cdot\bar{p}_x \cdot \left(\vec{X}_x^{(0)} \cdot \mathbf{v}_y\right) \\
&\quad= (1-\alpha)\cdot\bar{p}_x \cdot \vec{C}_x^{(0)}(y) \\
&\quad= (1-\alpha)\cdot\bar{p}_x \cdot \left(\frac{\alpha\cdot p_y}{(1-\alpha)\cdot\bar{p}_x} + p_y\right) \\
&\quad= (\alpha + (1-\alpha)\cdot\bar{p}_x)\cdot p_y \;=\; \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (0,1) \mid i^* \geq i\right].
\end{aligned}$$

$$\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (1,0) \mid i^* \geq i\right] \\
&\quad= (1-\alpha)\cdot p_x \cdot \sum_{\bar{x}:\; f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}_x^{(1)}}[x' = \bar{x}] \\
&\quad= (1-\alpha)\cdot p_x \cdot \left(1 - \vec{X}_x^{(1)} \cdot \mathbf{v}_y\right) \\
&\quad= (1-\alpha)\cdot p_x \cdot \left(1 - \vec{C}_x^{(1)}(y)\right) \\
&\quad= (1-\alpha)\cdot p_x \cdot (1 - p_y) \\
&\quad= (1-\alpha)\cdot p_x \cdot \bar{p}_y \;=\; \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (1,0) \mid i^* \geq i\right].
\end{aligned}$$

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (1,1) \mid i^* \geq i\right] \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=1} \Pr_{x' \leftarrow \vec{X}^{(1)}_x}[x' = \bar{x}] \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \left(\vec{X}^{(1)}_x \cdot \mathbf{v}_y\right) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \vec{C}^{(1)}_x(y) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot p_y \quad = \quad \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (1,1) \mid i^* \geq i\right].
\end{aligned}
$$

Equality holds, in all cases, between the corresponding probabilities in the ideal and hybrid worlds. We thus conclude that Eq. (3.19) holds for all x, y with $f(x,y) = 0$.

Case 2: x and y are such that $f(x,y) = 1$. We provide the calculations with limited discussion. In the hybrid world, we have

$$
\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (a,b) \mid i^* \geq i\right] = \begin{cases} ((1-\alpha)\cdot \bar{p}_x)\cdot \bar{p}_y & (a,b) = (0,0) \\ ((1-\alpha)\cdot \bar{p}_x)\cdot p_y & (a,b) = (0,1) \\ (\alpha + (1-\alpha)\cdot p_x)\cdot \bar{p}_y & (a,b) = (1,0) \\ (\alpha + (1-\alpha)\cdot p_x)\cdot p_y & (a,b) = (1,1) \end{cases}
$$

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x,y) = 1$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 0] = \bar{p}_x$, and the value of $\text{OUT}_{\mathsf{ideal}}$ is now dependent on the value of a_i. Working out the details, we have:

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,0) \mid i^* \geq i\right] \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}^{(0)}_x}[x' = \bar{x}] \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \left(1 - \vec{X}^{(0)}_x \cdot \mathbf{v}_y\right) \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \left(1 - \vec{C}^{(0)}_x(y)\right) \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \bar{p}_y.
\end{aligned}
$$

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,1) \mid i^* \geq i\right] \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=1} \Pr_{x' \leftarrow \vec{X}^{(0)}_x}[x' = \bar{x}] \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \left(\vec{X}^{(0)}_x \cdot \mathbf{v}_y\right) \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \left(\vec{C}^{(0)}_x(y)\right) \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot p_y.
\end{aligned}
$$

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (1,0) \mid i^* \geq i\right] \\
&\quad = (1-\alpha)\cdot p_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}_x^{(1)}}[x' = \bar{x}] \\
&\quad = (1-\alpha)\cdot p_x \cdot \left(1 - \vec{X}_x^{(1)} \cdot \mathbf{v}_y\right) \\
&\quad = (1-\alpha)\cdot p_x \cdot \left(1 - \vec{C}_x^{(1)}(y)\right) \\
&\quad = (1-\alpha)\cdot p_x \cdot \left(1 - \frac{\alpha\cdot(p_y-1)}{(1-\alpha)\cdot p_x} - p_y\right) \\
&\quad = (\alpha + (1-\alpha)\cdot p_x)\cdot \bar{p}_y.
\end{aligned}
$$

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (1,1) \mid i^* \geq i\right] \\
&\quad = \alpha + (1-\alpha)\cdot p_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=1} \Pr_{x' \leftarrow \vec{X}_x^{(1)}}[x' = \bar{x}] \\
&\quad = \alpha + (1-\alpha)\cdot p_x \cdot \left(\vec{X}_x^{(1)} \cdot \mathbf{v}_y\right) \\
&\quad = \alpha + (1-\alpha)\cdot p_x \cdot \left(\vec{C}_x^{(1)}(y)\right) \\
&\quad = \alpha + (1-\alpha)\cdot p_x \cdot \left(\frac{\alpha\cdot(p_y-1)}{(1-\alpha)\cdot p_x} + p_y\right) \\
&\quad = (\alpha + (1-\alpha)\cdot p_x)\cdot p_y.
\end{aligned}
$$

Once again, equality holds between the corresponding probabilities in the ideal and hybrid worlds in all cases. This concludes the proof of the theorem. ■

3.3 A Lower Bound for Functions Containing an Embedded XOR

In the previous section we have shown a protocol that enables completely fair computation of certain functions that contain an embedded XOR. That protocol, however, has round complexity $\omega(\log \kappa)$. (The round complexity may be worse, depending on α, but if α is constant then the round complexity is $m = \omega(\log \kappa)$.) In this section we prove that this is inherent for any function that has an embedded XOR.

3.3.1 Preliminaries

Let f be a single-output, boolean function with an embedded XOR; that is, a function for which there exist inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$. Let Π be an $r(\kappa)$-round protocol that securely computes f with complete fairness. Here we denote the two parties executing the protocol by A and B. We present some basic conventions below, as well as the specification of a series of fail-stop adversaries that we will use in our proof.

Notation and conventions: We assume that A sends the first message in protocol Π, and B sends the last message. A *round* of Π consists of a message from A followed by a message from B. If A aborts before sending its ith-round message (but after sending the first $i-1$ messages), then we denote by b_{i-1} the value output by B (so B outputs b_0 if A sends nothing). If B aborts before sending its ith-round message (but after sending the first $i-1$ messages),

then we denote by a_i the value output by A (so A outputs a_1 if B sends nothing). If neither party aborts, then B outputs b_r and A outputs a_{r+1}.

Proof overview. We consider executions of Π in which each party begins with input distributed uniformly in $\{x_0, x_1\}$ or $\{y_0, y_1\}$, respectively. We describe a series of $4r$ fail-stop adversaries $\{A_{i1}, A_{i0}, B_{i1}, B_{i0}\}_{i=1}^{r}$ where, intuitively, the aim of adversary A_{ib} is to guess B's input while *simultaneously* biasing B's output toward the bit b. (The aim of adversary B_{ib} is exactly analogous.) We show that if $r = O(\log \kappa)$, then one of these adversaries succeeds with "high" probability even though, as explained next, this is not possible in the ideal world.

In the ideal world evaluation of f (when B chooses its input at random in $\{y_0, y_1\}$), it is certainly possible for an adversary corrupting A to learn B's input with certainty (this follows from the fact that f contains an embedded XOR), and it may be possible, depending on f, to bias B's output with certainty. It is *not* possible, however, to do both simultaneously with high probability. (We formally state and prove this below.) This gives us our desired contradiction whenever $r = O(\log \kappa)$, and shows that no protocol with this many rounds can be completely fair.

Descriptions of the adversaries. Before giving the formal specification of the adversaries, we provide an intuitive description of adversary A_{i1}. (The other adversaries rely on the same intuition.) A_{i1} chooses a random input $x \in \{x_0, x_1\}$ and runs the protocol honestly for $i - 1$ rounds. It then computes the value it would output if B aborted the protocol at the current point, i.e., it computes a_i. If $a_i = 1$, then A_{i1} continues the protocol for one more round (hoping that this will cause B to output 1 also) and halts. If $a_i = 0$, then A_{i1} halts immediately (hoping that B's output does not yet "match" A_{i1}'s, and that B will still output 1). In addition to this behavior during the protocol, A_{i1} also guesses B's input, in the natural way, based on its own input value x and the value of a_i it computed. In particular, if $x = x_\sigma$ then A_{i1} guesses that B's input is $y_{a_i \oplus \sigma}$ (since $f(x_\sigma, y_{a_i \oplus \sigma}) = a_i$).

Say B's input is y. Intuitively, because the protocol is completely fair, if the output that A_{i1} computes in round i is biased toward the correct value of $f(x, y)$, it must be that the last message sent by A_{i1} has relatively limited relevance (i.e., that B would output the same bit whether A_{i1} sends its ith round message or not). In particular, in the case of A_{r1}, the computed output must be equal to $f(x, y)$ (with all but negligible probability), and therefore the last message of the protocol is, in some sense, unnecessary. Using induction (for a logarithmic number of steps) we will demonstrate that the same holds for each of the prior rounds, and conclude that a protocol running in $\mathcal{O}(\log \kappa)$ rounds can be transformed into an empty protocol in which neither party sends anything. This is, of course, impossible; therefore, no such protocol exists.

We now formally describe the adversaries.

Adversary A_{i1}:

1. Choose $x \in_R \{x_0, x_1\}$.
2. Run the honest A for the first $i - 1$ rounds (using input x) and compute a_i:
   (a) If $a_i = 1$ and $x = x_0$, then output $\mathsf{guess}(y = y_1)$, send the ith round message, and halt.
   (b) If $a_i = 1$ and $x = x_1$, then output $\mathsf{guess}(y = y_0)$, send the ith round message, and halt.
   (c) If $a_i = 0$ and $x = x_0$, then output $\mathsf{guess}(y = y_0)$ and halt immediately.
   (d) If $a_i = 0$ and $x = x_1$, then output $\mathsf{guess}(y = y_1)$ and halt immediately.

Adversary A_{i0}:

1. Choose $x \in_R \{x_0, x_1\}$.
2. Run the honest A for the first $i-1$ rounds (using input x) and compute a_i:
   (a) If $a_i = 0$ and $x = x_0$, then output guess$(y = y_0)$, send the ith round message and halt.
   (b) If $a_i = 0$ and $x = x_1$, then output guess$(y = y_1)$, send the ith round message and halt.
   (c) If $a_i = 1$ and $x = x_0$, then output guess$(y = y_1)$ and halt immediately.
   (d) If $a_i = 1$ and $x = x_1$, then output guess$(y = y_0)$ and halt immediately.

Adversary B_{i1}:

1. Choose $y \in_R \{y_0, y_1\}$.
2. Run the honest B for the first $i-1$ rounds (using input y), receive A's ith round message, and compute b_i:
   (a) If $b_i = 1$ and $y = y_0$, then output guess$(x = x_1)$, send the ith round message, and halt.
   (b) If $b_i = 1$ and $y = y_1$, then output guess$(x = x_0)$, send the ith round message, and halt.
   (c) If $b_i = 0$ and $y = y_0$, then output guess$(x = x_0)$ and halt immediately.
   (d) If $b_i = 0$ and $y = y_1$, then output guess$(x = x_1)$ and halt immediately.

Adversary B_{i0}:

1. Choose $y \in_R \{y_0, y_1\}$.
2. Run the honest B for the first $i-1$ rounds (using input y), receive A's ith round message, and compute b_i:
   (a) If $b_i = 0$ and $y = y_0$, then output guess$(x = x_0)$, send the ith round message, and halt.
   (b) If $b_i = 0$ and $y = y_1$, then output guess$(x = x_1)$, send the ith round message, and halt.
   (c) If $b_i = 1$ and $y = y_0$, then output guess$(x = x_1)$ and halt immediately.
   (d) If $b_i = 1$ and $y = y_1$, then output guess$(X = x_0)$ and halt immediately.

Success probability for A_{i1}: As preparation for the proof that follows, we calculate the probability that A_{i1} succeeds in *simultaneously* guessing B's input y correctly, and having B output 1. By construction, if (say) A_{i1} uses $x = x_0$ as input and obtains $a_i = 0$, then it guesses correctly iff $y = y_0$. Furthermore, since it received $a_i = 0$ it does not send its ith round message; thus, by our notation, B outputs 1 if $b_{i-1} = 1$. There are three other possible ways for this to occur as well:

$$\begin{aligned}
&\Pr[A_{i1} \text{ guesses } y \wedge B \text{ outputs } 1] \\
&= \Pr[x = x_0 \wedge y = y_0 \wedge a_i = 0 \wedge b_{i-1} = 1] + \Pr[x = x_0 \wedge y = y_1 \wedge a_i = 1 \wedge b_i = 1] \\
&\quad + \Pr[x = x_1 \wedge y = y_1 \wedge a_i = 0 \wedge b_{i-1} = 1] + \Pr[x = x_1 \wedge y = y_0 \wedge a_i = 1 \wedge b_i = 1].
\end{aligned}$$

The calculations are similar for A_{i0}, B_{i1}, and B_{i0} so we present them with no further explanation.

Success probability for A_{i0}:

$$\begin{aligned}&\Pr[A_{i0} \text{ guesses } y \wedge B \text{ outputs } 0]\\ &= \Pr[x = x_0 \wedge y = y_0 \wedge a_i = 0 \wedge b_i = 0] + \Pr[x = x_0 \wedge y = y_1 \wedge a_i = 1 \wedge b_{i-1} = 0]\\ &\quad + \Pr[x = x_1 \wedge y = y_1 \wedge a_i = 0 \wedge b_i = 0] + \Pr[x = x_1 \wedge y = y_0 \wedge a_i = 1 \wedge b_{i-1} = 0].\end{aligned}$$

Success probability for B_{i1}:

$$\begin{aligned}&\Pr[B_{i1} \text{ guesses } x \wedge A \text{ outputs } 1]\\ &= \Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_i = 1] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_{i+1} = 1]\\ &\quad + \Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_i = 1] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_{i+1} = 1].\end{aligned}$$

Success probability for B_{i0}:

$$\begin{aligned}&\Pr[B_{i0} \text{ guesses } x \wedge A \text{ outputs } 0]\\ &= \Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0]\\ &\quad + \Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0].\end{aligned}$$

3.3.2 The Proof

We begin by showing that, in the ideal model, it is impossible for an adversary to bias the output of the honest party while simultaneously guessing the honest party's input, with probability greater than $1/2$. Note that an adversary can certainly do one or the other. For example, if the honest B uses input $y \in_R \{y_0, y_1\}$ and an adversarial A uses input x_0, then A learns the input of B (by observing if the output is 0 or 1). Furthermore, if there exists a value x' for which $f(x', y_0) = f(x', y_1) = 1$ then A can completely bias the output of B to be 1.[4] In the first case, however, B's output is a random bit; in the second case, A learns no information about B's input. The following claim proves that these two extremes represent, in some sense, the best possible strategies:

Claim 3.3.1 *Consider an ideal-world evaluation of f (with complete fairness), where the honest party B chooses its input y uniformly from $\{y_0, y_1\}$ and the corrupted A^* outputs a guess for y following its interaction with the trusted party. For any A^* and any $\sigma \in \{0, 1\}$, it holds that*

$$\Pr[A^* \textit{ guesses } y \wedge B \textit{ outputs } \sigma] \leq \frac{1}{2}\,.$$

An analogous claim holds for the case when A is honest.

Proof: We consider the case of an honest B. Let $X_0 \stackrel{\text{def}}{=} \{x \mid f(x, y_0) = f(x, y_1) = 0\}$, and likewise $X_1 \stackrel{\text{def}}{=} \{x \mid f(x, y_0) = f(x, y_1) = 1\}$. Let $X_\oplus = \{x \mid f(x, y_0) \neq f(x, y_1)\}$. Note that X_0, X_1, and $X_\oplus$ partition the set of all inputs for A^*. In the following, when we say "A^*

[4] We stress that this is different from the case of boolean XOR, where it is impossible to bias the honest party's output at all in the ideal model (when the honest party uses a random input).

sends x" we mean that it sends x to the trusted party in the ideal model. For any $\sigma \in \{0,1\}$ we have:

$$\begin{aligned}
&\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma] \\
&= \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \wedge A^* \text{ sends } x \in X_{\bar{\sigma}}] \\
&\quad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \wedge A^* \text{ sends } x \in X_{\sigma}] \\
&\quad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \wedge A^* \text{ sends } x \in X_{\oplus}] \\
&= \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\bar{\sigma}}] \cdot \Pr[A^* \text{ sends } x \in X_{\bar{\sigma}}] \\
&\quad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\sigma}] \cdot \Pr[A^* \text{ sends } x \in X_{\sigma}] \\
&\quad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\oplus}] \cdot \Pr[A^* \text{ sends } x \in X_{\oplus}].
\end{aligned}$$

Clearly $\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\bar{\sigma}}] = 0$ since B *always* outputs $\bar{\sigma}$ when A^* sends $x \in X_{\bar{\sigma}}$. Also,

$$\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\sigma}] = \Pr[A^* \text{ guesses } y \mid A^* \text{ sends } x \in X_{\sigma}] = \frac{1}{2},$$

where the first equality is because party B *always* outputs σ when A^* sends $x \in X_{\sigma}$, and the second equality is because A learns no information about B's input (which was chosen uniformly from $\{y_0, y_1\}$). Finally,

$$\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } 1 \mid A^* \text{ sends } x \in X_{\oplus}] \leq \Pr[B \text{ outputs } 1 \mid A^* \text{ sends } x \in X_{\oplus}] = \frac{1}{2},$$

because B's input is chosen uniformly from $\{y_0, y_1\}$. Combining the above proves the claim. ■

The above claim, along with the assumed security of Π (with complete fairness), implies that for every inverse polynomial $\mu = 1/\mathsf{poly}$ we have

$$\Pr[B_{i0} \text{ guesses } x \wedge A \text{ outputs } 0] \leq \frac{1}{2} + \mu(\kappa) \tag{3.20}$$

$$\Pr[B_{i1} \text{ guesses } x \wedge A \text{ outputs } 1] \leq \frac{1}{2} + \mu(\kappa) \tag{3.21}$$

$$\Pr[A_{i0} \text{ guesses } y \wedge B \text{ outputs } 0] \leq \frac{1}{2} + \mu(\kappa) \tag{3.22}$$

$$\Pr[A_{i1} \text{ guesses } y \wedge B \text{ outputs } 1] \leq \frac{1}{2} + \mu(\kappa) \tag{3.23}$$

for sufficiently-large κ and all $1 \leq i \leq r(\kappa)$.

We now prove a claim that states, informally, that if both parties can compute the correct output with high probability after running i rounds of Π, then they can also compute the correct output with high probability even when B does not send its ith-round message.

Claim 3.3.2 *Fix a function μ and a value of κ for which Equations (3.20)–(3.23) hold for*

$1 \leq i \leq r(\kappa)$, *and let* $\mu = \mu(\kappa)$. *For any* $1 \leq i \leq r(\kappa)$, *if the following inequalities hold:*

$$\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_{i+1} = 1] - \frac{1}{4}\right| \leq \mu \tag{3.24}$$

$$\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_{i+1} = 1] - \frac{1}{4}\right| \leq \mu \tag{3.25}$$

$$\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_{i+1} = 0] - \frac{1}{4}\right| \leq \mu \tag{3.26}$$

$$\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_{i+1} = 0] - \frac{1}{4}\right| \leq \mu \tag{3.27}$$

when x *is chosen uniformly from* $\{x_0, x_1\}$ *and* y *is chosen uniformly from* $\{y_0, y_1\}$, *then:*

$$\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| \leq 4\mu \tag{3.28}$$

$$\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| \leq 4\mu \tag{3.29}$$

$$\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| \leq 4\mu \tag{3.30}$$

$$\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| \leq 4\mu \tag{3.31}$$

when x *and* y *are chosen in the same way.*

The first four equations represent the probability with which both parties receive correct output after executing the first i rounds of Π (i.e., after B sends its message in round i), for all possible choices of their inputs. The last four equations consider the same event, but when B does not send its message in round i. The claim asserts that the fact that B does not send its message in round i has a limited effect on the probability with which the parties obtain correct outputs.

Proof: We first prove Equation (3.28). That $\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] \leq \frac{1}{4} + 4\mu$ is immediate, since $\Pr[y = y_0 \wedge x = x_1] = \frac{1}{4}$. We must therefore prove the corresponding lower bound. Combining Equations (3.20), (3.26), and (3.27), and using our earlier calculation for the success probability for B_{i0}, we obtain

$$\begin{aligned}
\frac{1}{2} + \mu &\geq \Pr[B_{i0} \text{ guesses } x \wedge A \text{ outputs } 0] \\
&= \Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \\
&\quad + \Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0] \\
&\geq \frac{1}{4} - \mu + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \\
&\quad + \frac{1}{4} - \mu + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0] \\
&= \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0] \\
&\quad + \frac{1}{2} - 2\mu\,,
\end{aligned}$$

implying

$$\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \leq 3\mu\,. \tag{3.32}$$

We also have

$$\begin{aligned}\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] \\ = \quad \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1] \\ \geq \quad \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_{i+1} = 1] \geq \frac{1}{4} - \mu\,,\end{aligned}$$

using Equation (3.24) for the final inequality. Combined with (3.32), we conclude that

$$\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] \geq \frac{1}{4} - 4\mu,$$

proving Equation (3.28).

Using a symmetric argument, we can similarly prove Equation (3.29). Using an exactly analogous argument, but with adversary B_{i1} in place of B_{i0}, we can prove Equations (3.30) and (3.31). ■

The proof of the following claim exactly parallels the proof of the preceding claim, but using adversaries A_{i0} and A_{i1} instead of adversaries B_{i0} and B_{i1}.

Claim 3.3.3 *Fix a function μ and a value of κ for which Equations (3.20)–(3.23) hold for $1 \leq i \leq r(\kappa)$, and let $\mu = \mu(\kappa)$. For any $1 \leq i \leq r(\kappa)$, if the following inequalities hold:*

$$\begin{aligned}\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq \mu \\ \left|\Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq \mu \\ \left|\Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq \mu \\ \left|\Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq \mu\end{aligned}$$

when x is chosen uniformly from $\{x_0, x_1\}$ and y is chosen uniformly from $\{y_0, y_1\}$, then:

$$\begin{aligned}\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_{i-1} = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq 4\mu \\ \left|\Pr[y = y_1 \wedge x = x_0 \wedge b_{i-1} = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq 4\mu \\ \left|\Pr[y = y_0 \wedge x = x_0 \wedge b_{i-1} = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq 4\mu \\ \left|\Pr[y = y_1 \wedge x = x_1 \wedge b_{i-1} = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq 4\mu\end{aligned}$$

when x and y are chosen in the same way.

We now prove the following theorem.

Theorem 3.3.4 *Let f be a two-party function containing an embedded XOR. Then any protocol securely computing f with complete fairness (assuming one exists) requires $\omega(\log \kappa)$ rounds.*

Proof: Let Π be a protocol computing f with complete fairness using $r = r(\kappa)$ rounds. Set $\mu = 1/\mathsf{poly}(\kappa)$ for some polynomial to be fixed later. By correctness of Π, we have that for κ sufficiently large

$$\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_r = 1 \wedge a_{r+1} = 1] - \frac{1}{4}\right| \leq \mu(\kappa)$$
$$\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_r = 1 \wedge a_{r+1} = 1] - \frac{1}{4}\right| \leq \mu(\kappa)$$
$$\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_r = 0 \wedge a_{r+1} = 0] - \frac{1}{4}\right| \leq \mu(\kappa)$$
$$\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_r = 0 \wedge a_{r+1} = 0] - \frac{1}{4}\right| \leq \mu(\kappa)$$

when x and y are chosen uniformly from $\{x_0, x_1\}$ and $\{y_0, y_1\}$, respectively. Taking κ large enough so that Equations (3.20)–(3.23) also hold for $1 \leq i \leq r(\kappa)$, we see that Claim 3.3.2 may be applied with $i = r$. Since the conclusion of Claim 3.3.2 is the assumption of Claim 3.3.3 and vice versa, the claims can be repeatedly applied r times, yielding:

$$\begin{aligned}
\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_0 = 1 \wedge a_1 = 1] - \frac{1}{4}\right| &\leq 4^{2r(\kappa)} \cdot \mu(\kappa) \\
\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_0 = 1 \wedge a_1 = 1] - \frac{1}{4}\right| &\leq 4^{2r(\kappa)} \cdot \mu(\kappa) \\
\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_0 = 0 \wedge a_1 = 0] - \frac{1}{4}\right| &\leq 4^{2r(\kappa)} \cdot \mu(\kappa) \\
\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_0 = 0 \wedge a_1 = 0] - \frac{1}{4}\right| &\leq 4^{2r(\kappa)} \cdot \mu(\kappa)\,.
\end{aligned}$$

If $r = \mathcal{O}(\log \kappa)$, then $p(n) \stackrel{\text{def}}{=} 4^{2r(\kappa)}$ is polynomial. Taking $\mu(\kappa) = 1/16p(\kappa)$ implies that, for κ sufficiently large, A and B can both correctly compute (with probability at least $3/4$) the value $f(x, y)$, for all $x \in \{x_0, x_1\}$ and $y \in \{y_0, y_1\}$, *without any interaction at all*. This is impossible, and so we conclude that $r = \omega(\log \kappa)$. ■

Chapter 4

Complete Fairness in Secure Multi-Party Computation

4.1 Fair Computation of Majority for Three Players

In this section we describe a completely-fair protocol for computing the majority function, maj, over boolean inputs, for the case of $n = 3$ parties. The high-level structure of our protocol follows the protocols of the prior chapters. However, we need to make some natural changes to the protocols to extend them to the multi-party setting. In particular, ShareGen has to be modified to output secret shares of 3 sequences of bits. Notationally, then, instead of values a_i and b_i representing outputs in round i, we define 3 values, $b_1^{(i)}, b_2^{(i)}$ and $b_3^{(i)}$. Note, though, $b_j^{(i)}$ does *not* represent the output of P_j if the other players abort. Rather, it will denote the outputs of *both* players P_{j-1} and P_{j+1} (where addition is done mod 3) in case player P_j aborts in round i. The reason for this notational switch relates to the need for two honest players to agree on a single output when the third player aborts; this concern was not an issue in the two-party case. As before, the output of ShareGen is sequences of secret shares of these $b_j^{(i)}$ values, but now the secret shares are random three-way shares; any $b_j^{(i)}$ can only be reconstructed given all three shares, denoted by $b_{j|1}^{(i)}, b_{j|2}^{(i)}, b_{j|3}^{(i)}$. The values $b_j^{(i)}$ are computed probabilistically, in the same manner as they were in Section 3.2 (cf. Figure 3.3). That is, a round i^* is first chosen according to a geometric distribution with parameter $\alpha = 1/5$. (As before, we will set the round complexity m such that $i^* \leq m$ with all but negligible probability.) Then, for $i < i^*$ the value of $b_j^{(i)}$ is computed using the true inputs of P_{j-1} and P_{j+1} but a random input for P_j; for $i \geq i^*$ the value $b_j^{(i)}$ is set equal to the correct output (i.e., it is computed using the true inputs of all parties). Note that even an adversary who knows all the parties' inputs and learns, sequentially, the values (say) $b_1^{(1)}, b_1^{(2)}, \ldots$ cannot determine definitively when round i^* occurs. We choose the protocol π computing ShareGen to be secure-with-designated-abort for P_1, according to Definition 2.6.1

The second phase of the protocol proceeds in a sequence of $m = \omega(\log n)$ iterations. (See Figure 4.2.) In each iteration i, each party P_j broadcasts its share of $b_j^{(i)}$. (We stress that we allow rushing, and do not assume synchronous broadcast.) Observe that, after this is done, parties P_{j-1} and P_{j+1} *jointly* have enough information to reconstruct $b_j^{(i)}$, but neither party has any information about $b_j^{(i)}$ on its own. If all parties behave honestly until the end of the protocol, then in the final iteration all parties reconstruct $b_1^{(m)}$ and output this value. If a

ShareGen

Inputs: Let the inputs to ShareGen be $x_1, x_2, x_3 \in \{0,1\}$. (If one of the received inputs is not in the correct domain, then a default value of 1 is used for that player.)

Computation:

1. Define values $b_1^{(1)}, \ldots, b_1^{(m)}$, $b_2^{(1)}, \ldots, b_2^{(m)}$ and $b_3^{(1)}, \ldots, b_3^{(m)}$ in the following way:
   - Choose $i^* \geq 1$ according to a geometric distribution with parameter $\alpha = 1/5$ (see text).
   - For $i = 0$ to $i^* - 1$ and $j \in \{1,2,3\}$ do:
     - Choose $\hat{x}_j \leftarrow \{0,1\}$ at random.
     - Set $b_j^{(i)} = \mathsf{maj}(x_{j-1}, \hat{x}_j, x_{j+1})$.
   - For $i = i^*$ to m and $j \in \{1,2,3\}$, set $b_j^{(i)} = \mathsf{maj}(x_1, x_2, x_3)$.
2. For $0 \leq i \leq m$ and $j \in \{1,2,3\}$, choose $b_{j|1}^{(i)}$, $b_{j|2}^{(i)}$ and $b_{j|3}^{(i)}$ as random three-way shares of $b_j^{(i)}$. (E.g., $b_{j|1}^{(i)}$ and $b_{j|2}^{(i)}$ are random and $b_{j|3}^{(i)} = b_{j|1}^{(i)} \oplus b_{j|2}^{(i)} \oplus b_j^{(i)}$.)
3. Let $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$. For $0 \leq i \leq m$, and $j, j' \in \{1,2,3\}$, let $\sigma_{j|j'}^{(i)} = \mathsf{Sign}_{sk}(i \| j \| j' \| b_{j|j'}^{(i)})$.

Output:

1. Send to each P_j the public key pk and the values $\left\{(b_{1|j}^{(i)}, \sigma_{1|j}^{(i)}), (b_{2|j}^{(i)}, \sigma_{2|j}^{(i)}), (b_{3|j}^{(i)}, \sigma_{3|j}^{(i)})\right\}_{i=0}^{m}$. Additionally, for each $j \in \{1,2,3\}$ parties P_{j-1} and P_{j+1} receive the value $b_{j|j}^{(0)}$.

Figure 4.1: Functionality ShareGen.

single party P_j aborts in some iteration i, then the remaining players P_{j-1} and P_{j+1} jointly reconstruct the value $b_j^{(i-1)}$ and output this value. (Recall that these two parties jointly have enough information to do this.) If two parties abort in some iteration i (whether at the same time, or one after the other) then the remaining party simply outputs its own input.

One other technical difference in this protocol as compared to the general protocol given in Section 2.8 is that we have to switch from using information theoretic MACs to digital signatures. This is because we now have to worry about collusion. Suppose, for example, that P_{j+1} holds a key that allows him to verify the value $b_j^{(i)}$ sent by P_j. If they are both corrupt, P_{j+1} can simply give this key to P_j and allow him to forge MACs on incorrect values. Instead, we use digital signatures, which separate the ability to verify from the ability to sign (see Figure 4.1).

We refer to Figures 4.1 and 4.2 for the formal specification of the protocol. We now prove that this protocol securely computes maj with complete fairness.

Theorem 4.1.1 *Assume that* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a secure signature scheme, that* π *securely computes* ShareGen *with designated abort, and that* π_{OR} *securely computes* OR *with complete fairness.*[1] *Then the protocol in Figure 4.2 securely computes* maj *with complete fairness.*

Proof: Let Π_{maj} denote the protocol of Figure 4.2. Observe that Π_{maj} yields the correct output with all but negligible probability when all players are honest. This is because, with all but negligible probability, $i^* \leq m$, and then $b_j^{(m)} = \mathsf{maj}(x_1, x_2, x_3)$. We thus focus on security of Π_{maj}.

[1] It is shown in Chapter 3 that such a protocol exists under standard assumptions.

Π_{maj}

Inputs: Party P_i has input $x_i \in \{0,1\}$.

The protocol:

1. **Preliminary phase:**
   (a) Parties P_1, P_2 and P_3 run a protocol π for computing ShareGen. Each player uses their respective inputs, x_1, x_2 and x_3, and security parameter κ.
   (b) If P_2 and P_3 receive $\perp$ from this execution, then P_2 and P_3 run a two-party protocol π_{OR} to compute the logical-or of their inputs.
   Otherwise, continue to the next stage.

   In what follows, parties always verify signatures; invalid signatures are treated as an abort.

2. **For $i = 1, \ldots, m-1$ do:**
   Broadcast shares:
   (a) Each P_j broadcasts $(b^{(i)}_{j|j}, \sigma^{(i)}_{j|j})$.
   (b) If (only) P_j aborts:
      i. P_{j-1} and P_{j+1} broadcast $(b^{(i-1)}_{j|j-1}, \sigma^{(i-1)}_{j|j-1})$ and $(b^{(i-1)}_{j|j+1}, \sigma^{(i-1)}_{j|j+1})$, respectively.
      ii. If one of P_{j-1}, P_{j+1} aborts in the previous step, the remaining player outputs its own input value. Otherwise, P_{j-1} and P_{j+1} both output $b^{(i-1)}_j = b^{(i-1)}_{j|1} \oplus b^{(i-1)}_{j|2} \oplus b^{(i-1)}_{j|3}$. (Recall that if $i = 1$, parties P_{j-1} and P_{j+1} received $b^{(0)}_{j|j}$ as output from π.)
   (c) If two parties abort, the remaining player outputs its own input value.

3. **In round $i = m$ do:**
   (a) Each P_j broadcasts $b^{(m)}_{1|j}, \sigma^{(m)}_{1|j}$.
   (b) If no one aborts, then all players output $b^{(m)}_1 = b^{(m)}_{1|1} \oplus b^{(m)}_{1|2} \oplus b^{(m)}_{1|3}$. If (only) P_j aborts, then P_{j-1} and P_{j+1} proceed as in step 2b. If two players abort, the remaining player outputs its own input value as in step 2c.

Figure 4.2: A protocol for computing majority.

We note that when no parties are corrupt, the proof of security is straightforward, since we assume the existence of a private broadcast channel. We therefore consider separately the cases when a single party is corrupted and when two parties are corrupted. Since the entire protocol is symmetric except for the fact that P_1 may choose to abort π, without loss of generality we may analyze the case when the adversary corrupts P_1 and the case when the adversary corrupts $\{P_1, P_2\}$. In each case, we prove security of Π_{maj} in a hybrid world where there is an ideal functionality computing ShareGen (with abort) as well as an ideal functionality computing OR (with complete fairness). Applying the composition theorem of [20] then gives the desired result.

Claim 4.1.2 *For every non-uniform, poly-time adversary $\mathcal{A}$ corrupting P_1 and running Π_{maj} in a hybrid model with access to ideal functionalities computing* ShareGen *(with abort) and* OR *(with complete fairness), there exists a non-uniform, poly-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing* maj *(with complete fairness), such that*

$$\left\{\text{IDEAL}_{\mathsf{maj},\mathcal{S}}(x_1,x_2,x_3,\kappa)\right\}_{x_i\in\{0,1\},\kappa\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen},\mathsf{OR}}_{\Pi_{\mathsf{maj}},\mathcal{A}}(x_1,x_2,x_3,\kappa)\right\}_{x_i\in\{0,1\},\kappa\in\mathbb{N}}$$

Proof:

Fix some polynomial-time adversary $\mathcal{A}$ corrupting P_1. We now describe a simulator $\mathcal{S}$ that also corrupts P_1 and runs $\mathcal{A}$ as a black box.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input x_1, the auxiliary input z, and the security parameter κ.
2. $\mathcal{S}$ receives input $x_1' \in \{0,1\}$ on behalf of P_1 as input to ShareGen.
3. $\mathcal{S}$ computes $(sk, pk) \leftarrow \mathsf{Gen}(1^\kappa)$, and gives to $\mathcal{A}$ the public key pk and values $b_{2|2}^{(0)}$, $b_{3|3}^{(0)}$, and $\left\{b_{1|1}^{(i)}, b_{2|1}^{(i)}, b_{3|1}^{(i)}\right\}_{i=0}^{m}$ (along with their appropriate signatures) chosen uniformly at random.
4. If $\mathcal{A}$ aborts execution of ShareGen, then $\mathcal{S}$ sends 1 to the trusted party computing maj, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise, $\mathcal{S}$ picks a value i^* according to a geometric distribution with parameter $\alpha = \frac{1}{5}$.

   For simplicity in what follows, we ignore the presence of signatures and leave the following implicit from now on: (1) $\mathcal{S}$ always computes an appropriate signature when sending any value to $\mathcal{A}$; (2) $\mathcal{S}$ treats an incorrect signature as an abort; and (3) if $\mathcal{S}$ ever receives a valid signature on a previously unsigned message, then $\mathcal{S}$ outputs fail and halts.
5. $\mathcal{S}$ now simulates the rounds of the protocol one-by-one: for $i = 1$ to $m-1$, the simulator chooses random $b_{2|2}^{(i)}$ and $b_{3|3}^{(i)}$ and sends these to $\mathcal{A}$. During this step, an abort by $\mathcal{A}$ (on behalf of P_1) is treated as follows:

   (a) If P_1 aborts in round $i \leq i^*$, then $\mathcal{S}$ chooses a random value $\hat{x}_1$ and sends it to the trusted party computing maj.

   (b) If P_1 aborts in round $i > i^*$, then $\mathcal{S}$ submits x_1' to the trusted party computing maj.

   In either case, $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs and halts.
6. If P_1 has not yet aborted, $\mathcal{S}$ then simulates the final round of the protocol. $\mathcal{S}$ sends x_1' to the trusted party, receives $b_{\mathsf{out}} = \mathsf{maj}(x_1', x_2, x_3)$, and chooses $b_{1|2}^{(m)}$ and $b_{1|3}^{(m)}$ at random subject to $b_{1|2}^{(m)} \oplus b_{1|3}^{(m)} \oplus b_{1|1}^{(m)} = b_{\mathsf{out}}$. $\mathcal{S}$ then gives these values to $\mathcal{A}$, outputs whatever $\mathcal{A}$ outputs, and halts.

Due to the security of the underlying signature scheme, the probability that $\mathcal{S}$ outputs fail is negligible in κ. Note that the view of P_1 is otherwise statistically close in both worlds. Indeed, until round m the view of P_1 is independent of the inputs of the other parties in both the real and ideal worlds. In round m itself, P_1 learns the (correct) output b_{out} in the ideal world and learns this value with all but negligible probability in the real world.

We therefore only have to argue that outputs of the two honest parties in the real and ideal worlds are statistically close. Clearly this is true if P_1 never aborts. As for the case when P_1 aborts at some point during the protocol, we divide our analysis into the following cases:

- If P_1 aborts the execution of ShareGen in step 4, then $\mathcal{S}$ submits '1' on behalf of P_1 to the trusted party computing maj. Thus, in the ideal world, the outputs of P_2 and P_3 will be $\mathsf{maj}(1, x_2, x_3)$. In the real world, if P_1 aborts computation of ShareGen, the honest parties output $\mathsf{OR}(x_2, x_3)$. Since $\mathsf{maj}(1, x_2, x_3) = \mathsf{OR}(x_2, x_3)$, their outputs are the same.

- If P_1 aborts in round i of the protocol (cf. step 5), then in both the real and ideal worlds the following holds:
  - If $i \leq i^*$, then P_2 and P_3 output $\mathsf{maj}(\hat{x}_1, x_2, x_3)$ where $\hat{x}_1$ is chosen uniformly at random.
  - If $i > i^*$, then P_2 and P_3 output $\mathsf{maj}(x'_1, x_2, x_3)$

  Since i^* is identically distributed in both worlds, the outputs of P_2 and P_3 in this case are identically distributed as well.
- If P_1 aborts in round m (cf. step 6), then in the ideal world the honest parties will output $\mathsf{maj}(x'_1, x_2, x_3)$. In the real world the honest parties output $\mathsf{maj}(x'_1, x_2, x_3)$ as long as $i^* \leq m - 1$, which occurs with all but negligible probability.

This completes the proof. ■

Claim 4.1.3 *For every non-uniform, poly-time adversary $\mathcal{A}$ corrupting P_1 and P_2 and running Π_{maj} in a hybrid model with access to ideal functionalities computing* ShareGen *(with abort) and* OR *(with completes fairness), there exists a non-uniform, poly-time adversary $\mathcal{S}$ corrupting P_1 and P_2 and running in the ideal world with access to an ideal functionality computing* maj *(with complete fairness), such that*

$$\left\{\text{IDEAL}_{\mathsf{maj},\mathcal{S}}(x_1, x_2, x_3, \kappa)\right\}_{x_i \in \{0,1\}, \kappa \in \mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen},\mathsf{OR}}_{\Pi_{\mathsf{maj}},\mathcal{A}}(x_1, x_2, x_3, \kappa)\right\}_{x_i \in \{0,1\}, \kappa \in \mathbb{N}}.$$

Proof: This case is significantly more complex than the case when only a single party is corrupted, since here $\mathcal{A}$ learns $b_3^{(i)}$ in each iteration i of the second phase. As in Section 3.2.1, we must deal with the fact that $\mathcal{A}$ might abort exactly in iteration i^*, after learning the correct output but before P_3 has enough information to compute the correct output.

We now describe a simulator $\mathcal{S}$ who corrupts P_1 and P_2 and runs $\mathcal{A}$ as a black-box. For ease of exposition in what follows, we sometimes refer to the actions of P_1 and P_2 when more formally we mean the action of $\mathcal{A}$ on behalf of those parties.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the inputs x_1 and x_2, the auxiliary input z, and the security parameter κ.
2. $\mathcal{S}$ receives x'_1 and x'_2 from P_1 and P_2, respectively, as input to ShareGen. If $x'_1 \notin \{0,1\}$ (resp., $x'_2 \notin \{0,1\}$), then $\mathcal{S}$ sets $x'_1 = 1$ (resp., $x'_2 = 1$).
3. $\mathcal{S}$ computes $(sk, pk) \leftarrow \mathsf{Gen}(1^\kappa)$, and then generates shares as follows:
   (a) Choose $\left\{b^{(i)}_{1|1}, b^{(i)}_{2|1}, b^{(i)}_{3|1}, b^{(i)}_{1|2}, b^{(i)}_{2|2}, b^{(i)}_{3|2}\right\}_{i=0}^{m}$ uniformly at random.
   (b) Choose $\hat{x}_3 \leftarrow \{0,1\}$ and set $b_3^{(0)} = \mathsf{maj}(x'_1, x'_2, \hat{x}_3)$. Set $b^{(0)}_{3|3} = b_3^{(0)} \oplus b^{(0)}_{3|1} \oplus b^{(0)}_{3|2}$.

   $\mathcal{S}$ then hands $\mathcal{A}$ the public key pk, the values $\left\{b^{(i)}_{1|1}, b^{(i)}_{2|1}, b^{(i)}_{3|1}, b^{(i)}_{1|2}, b^{(i)}_{2|2}, b^{(i)}_{3|2}\right\}_{i=0}^{m}$ (along with their appropriate signatures), and the value $b^{(0)}_{3|3}$ as the outputs of P_1 and P_2 from ShareGen.
4. If P_1 aborts execution of ShareGen, then $\mathcal{S}$ extracts x''_2 from P_2 as its input to OR. It then sends $(1, x''_2)$ to the trusted party computing maj, outputs whatever $\mathcal{A}$ outputs, and halts.

5. Otherwise, if P_1 does not abort, then $\mathcal{S}$ picks a value i^* according to a geometric distribution with parameter $\alpha = \frac{1}{5}$.

   In what follows, for ease of description, we will use x_1 and x_2 in place of x'_1 and x'_2, keeping in mind that that $\mathcal{A}$ could of course have used substituted inputs. We also ignore the presence of signatures from now on, and leave the following implicit in what follows: (1) $\mathcal{S}$ always computes an appropriate signature when sending any value to $\mathcal{A}$; (2) $\mathcal{S}$ treats an incorrect signature as an abort; and (3) if $\mathcal{S}$ ever receives a valid signature on a previously unsigned message (i.e., a *forgery*), then $\mathcal{S}$ outputs fail and halts.

   Also, from here on we will say that $\mathcal{S}$ *sends* b *to* $\mathcal{A}$ *in round* i if $\mathcal{S}$ sends a value $b^{(i)}_{3|3}$ such that $b^{(i)}_{3|3} \oplus b^{(i)}_{3|1} \oplus b^{(i)}_{3|2} = b^{(i)}_3 = b$.

6. For round $i = 1, \ldots, i^* - 1$, the simulator $\mathcal{S}$ computes and then sends $b^{(i)}_3$ as follows:

   (a) Select $\hat{x}_3 \leftarrow \{0,1\}$ at random.

   (b) $b^{(i)}_3 = \mathsf{maj}(x_1, x_2, \hat{x}_3)$.

7. If P_1 aborts in round $i < i^*$, then $\mathcal{S}$ sets $\hat{x}_2 = x_2$ and assigns a value to $\hat{x}_1$ according to the following rules that depend on the values of (x_1, x_2) and on the value of $b^{(i)}_3$:

   (a) If $x_1 = x_2$, then $\mathcal{S}$ sets $\hat{x}_1 = x_1$ with probability $\frac{3}{8}$ (and sets $\hat{x}_1 = \bar{x}_1$ otherwise).

   (b) If $x_1 \neq x_2$ and $b^{(i)}_3 = x_1$, then $\mathcal{S}$ sets $\hat{x}_1 = x_1$ with probability $\frac{1}{4}$ (and sets $\hat{x}_1 = \bar{x}_1$ otherwise).

   (c) If $x_1 \neq x_2$ and $b^{(i)}_3 = x_2$, then $\mathcal{S}$ sets $\hat{x}_1 = x_1$ with probability $\frac{1}{2}$ (and sets $\hat{x}_1 = \bar{x}_1$ otherwise).

   $\mathcal{S}$ then finishes the simulation as follows:

   (a) If $\hat{x}_1 \neq \hat{x}_2$, then $\mathcal{S}$ submits $(\hat{x}_1, \hat{x}_2)$ to the trusted party computing maj. Denote the output it receives from the trusted party by b_{out}. Then $\mathcal{S}$ sets $b^{(i-1)}_1 = b_{\mathsf{out}}$, computes $b^{(i-1)}_{1|3} = b^{(i-1)}_1 \oplus b^{(i-1)}_{1|1} \oplus b^{(i-1)}_{1|2}$, sends $b^{(i-1)}_{1|3}$ to P_2 (on behalf of P_3), outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) If $\hat{x}_1 = \hat{x}_2$, then $\mathcal{S}$ sets $b^{(i-1)}_1 = \hat{x}_1 = \hat{x}_2$, computes $b^{(i-1)}_{1|3} = b^{(i-1)}_1 \oplus b^{(i-1)}_{1|1} \oplus b^{(i-1)}_{1|2}$, and sends $b^{(i-1)}_{1|3}$ to P_2 (on behalf of P_3). (We stress that this is done *before* sending anything to the trusted party computing maj.) If P_2 aborts, then $\mathcal{S}$ sends $(0,1)$ to the trusted party computing maj. Otherwise, it sends $(\hat{x}_1, \hat{x}_2)$ to the trusted party computing maj. In both cases it outputs whatever $\mathcal{A}$ outputs, and then halts.

   If P_2 aborts in round $i < i^*$, then $\mathcal{S}$ acts analogously but swapping the roles of P_1 and P_2 as well as x_1 and x_2.

   If both parties abort simultaneously in round $i < i^*$, then $\mathcal{S}$ sends $(0,1)$ to the trusted party computing maj, outputs whatever $\mathcal{A}$ outputs, and halts.

8. In round i^*:

   (a) If $x_1 \neq x_2$, then $\mathcal{S}$ submits (x_1, x_2) to the trusted party. Let $b_{\mathsf{out}} = \mathsf{maj}(x_1, x_2, x_3)$ denote the output.

(b) If $x_1 = x_2$, then $\mathcal{S}$ simply sets $b_{\mathsf{out}} = x_1 = x_2$ *without querying the trusted party* and continues. (Note that in this case, $b_{\mathsf{out}} = \mathsf{maj}(x_1, x_2, x_3)$ even though $\mathcal{S}$ did not query the trusted party.)

9. In rounds $i^*, \ldots, m-1$, the simulator $\mathcal{S}$ sends b_{out} to $\mathcal{A}$.

   If $\mathcal{A}$ aborts P_1 and P_2 simultaneously, then $\mathcal{S}$ submits $(1,0)$ to the trusted party (if he hasn't already done so in step 8a), outputs whatever $\mathcal{A}$ outputs, and halts.

   If $\mathcal{A}$ aborts P_1 (only), then $\mathcal{S}$ sets $b_1^{(i-1)} = b_{\mathsf{out}}$, computes $b_{1|3}^{(i-1)} = b_1^{(i-1)} \oplus b_{1|1}^{(i-1)} \oplus b_{1|2}^{(i-1)}$, and sends $b_{1|3}^{(i-1)}$ to P_2 (on behalf of P_3). Then:

   Case 1: $x_1 \neq x_2$. Here $\mathcal{S}$ has already sent (x_1, x_2) to the trusted party. So $\mathcal{S}$ simply outputs whatever $\mathcal{A}$ outputs and ends the simulation.

   Case 2: $x_1 = x_2$. If P_2 does not abort, then $\mathcal{S}$ sends (x_1, x_2) to the trusted party. If P_2 aborts, then $\mathcal{S}$ sends $(0,1)$ to the trusted party. In both cases $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs and halts.

   If $\mathcal{A}$ aborts P_2 (only), then $\mathcal{S}$ acts as above but swapping the roles of P_1, P_2 and x_1, x_2. If $\mathcal{A}$ does not abort anyone through round m, then $\mathcal{S}$ sends (x_1, x_2) to the trusted party (if he hasn't already done so), outputs what $\mathcal{A}$ outputs, and halts.

We first note that the probability $\mathcal{S}$ outputs fail is negligible, due to the security of the underling signature scheme. We state the following claim:

Claim 4.1.4 *If P_1 and P_2 both abort, then $\mathcal{S}$ always sends $(0,1)$ or $(1,0)$ to the trusted party.*

We leave verification to the reader. We must prove that for any set of inputs, the joint distribution over the possible views of $\mathcal{A}$ and the output of P_3 is equal in the ideal and hybrid worlds:

$$(\text{VIEW}_{\mathsf{hyb}}(x_1, x_2, x_3), \text{OUT}_{\mathsf{hyb}}(x_1, x_2, x_3)) \equiv (\text{VIEW}_{\mathsf{ideal}}(x_1, x_2, x_3), \text{OUT}_{\mathsf{ideal}}(x_1, x_2, x_3)) \quad (4.1)$$

We begin by noting that this is trivially true when no players ever abort. It is also easy to verify that this is true when P_1 aborts during the execution of ShareGen. From here forward, we therefore assume that $\mathcal{A}$ aborts player P_1 at some point after the execution of ShareGen. We consider what happens when $\mathcal{A}$ aborts P_2 as well, but for simplicity we will only analyze the cases where P_1 is aborted first, and when they are aborted at the same time. The analysis when $\mathcal{A}$ aborts only P_2, or when he aborts P_1 after P_2 is symmetric and is not dealt with here. We will break up the view of $\mathcal{A}$ into two parts: the view before P_1 aborts, where a particular instance of this view is denoted by $\vec{a}_i$, and the single message intended for P_2 that $\mathcal{A}$ receives after P_1 aborts, denoted by $b_1^{(i-1)}$. Letting i denote the round in which P_1 aborts, and b_{out} the value output by P_3, we wish to prove:

$$\Pr\left[(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}})\right] = \Pr\left[(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}})\right]$$

(we drop explicit mention of the inputs to improve readability). Towards proving this, we first prove the following two claims.

Claim 4.1.5 *For all inputs and all feasible adversarial views* $(\vec{a}_i, b_1^{(i-1)})$,

$$\begin{aligned}&\Pr\left[(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}}) \bigwedge i > i^*\right] \\ &= \Pr\left[(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}}) \bigwedge i > i^*\right]\end{aligned}$$

Proof: We denote by $P_2^{\perp}$ the event that P_2 aborts the protocol (either at the same time as P_1, or after P_1 aborts, during the exchange of the shares of $b_1^{(i-1)}$). We also denote the event $\left(\text{VIEW}_{\mathsf{hyb}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right)$ by E_{hyb}, and the event $\left(\text{VIEW}_{\mathsf{ideal}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right)$ by $\text{E}_{\mathsf{ideal}}$ in order to shorten notation. We have the following:

$$\begin{aligned}&\Pr\left[(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}}) \bigwedge i > i^*\right] \\ &= Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \bigwedge P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] + \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \bigwedge \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \\ &= \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \\ &\quad + \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[\neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right]\end{aligned}$$

and the same is true in the ideal world. It follows from the descriptions of the protocol and the simulator that

$$\Pr\left[P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{hyb}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right] = \Pr\left[P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{ideal}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right]$$

and similarly that

$$\Pr\left[\neg P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{hyb}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right] = \Pr\left[\neg P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{ideal}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right].$$

The above two equalities hold because the protocol is designed such that any view $\vec{a}_i$ occurs with the same probability in both worlds. Furthermore, given that $i > i^*$, it holds that $b_1^{(i-1)} = f(x_1, x_2, x_3)$, independent of $\vec{a}_i$. P_2 decides whether to abort based only on these two variables, so the decision is the same in both worlds. We therefore need only to prove that

$$\begin{aligned}&\Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{hyb}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right] \\ &= \Pr\left[\text{OUT}_{\mathsf{ideal}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{ideal}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right] \qquad (4.2)\end{aligned}$$

and that

$$\begin{aligned}&\Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{hyb}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right] \\ &= \Pr\left[\text{OUT}_{\mathsf{ideal}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{VIEW}_{\mathsf{ideal}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i > i^*\right] \qquad (4.3)\end{aligned}$$

Both equations follow easily again from the protocol and simulator descriptions. To see Equation 4.2, note that in the hybrid world when both P_1 and P_2 abort, P_3 always outputs his own input, $b_{\mathsf{out}} = x_3$. In the ideal world, recall from claim 4.1.4 that anytime P_1 and P_2 both abort, and in particular in round $i > i^*$, $\mathcal{S}$ submits either $(0, 1)$ or $(1, 0)$ to the trusted party, resulting in $b_{\mathsf{out}} = x_3$. For Equation 4.3, note that in the hybrid world when P_1 aborts in round $i > i^*$, and P_2 does not, P_3 outputs $b_{\mathsf{out}} = f(x_1, x_2, x_3)$. In the ideal world, this is also true, as $\mathcal{S}$

submits (x_1, x_2) to the trusted party (either in step 8a or in step 9). ■

We proceed now to the more difficult claim, in the case when $i \leq i^*$:

Claim 4.1.6 *For all inputs, for all outputs b_{out}, and for all feasible adversarial views $(\vec{a}_i, b_1^{(i-1)})$,*

$$\begin{aligned}&\Pr\left[(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}}) \bigwedge i \leq i^*\right] \\ &= \Pr\left[(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}}) \bigwedge i \leq i^*\right]\end{aligned}$$

Proof: We denote by $P_2^{\perp}$, as before, the event that P_2 aborts the protocol. We now replace the event $\left(\text{VIEW}_{\mathsf{hyb}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i \leq i^*\right)$ by E_{hyb}, and the event $\left(\text{VIEW}_{\mathsf{ideal}} = (\vec{a}_i, b_1^{(i-1)}) \bigwedge i \leq i^*\right)$ by $\text{E}_{\mathsf{ideal}}$ to shorten notation. We again have that

$$\begin{aligned}&\Pr\left[(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}) = (\vec{a}_i, b_1^{(i-1)}, b_{\mathsf{out}}) \bigwedge i \leq i^*\right] \\ &= \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \bigwedge P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] + \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \bigwedge \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \\ &= \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \\ &\quad + \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[\neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right]\end{aligned}$$

and again, the same probabilistic argument holds in the ideal world. Rewriting the above, therefore, we equivalently must prove that

$$\begin{aligned}&\Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[P_2^{\perp} \mid \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[\text{E}_{\mathsf{hyb}}\right] \\ &\quad + \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[\neg P_2^{\perp} \mid \text{E}_{\mathsf{hyb}}\right] \cdot \Pr\left[\text{E}_{\mathsf{hyb}}\right] \\ &= \Pr\left[\text{OUT}_{\mathsf{ideal}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{E}_{\mathsf{ideal}}\right] \cdot \Pr\left[P_2^{\perp} \mid \text{E}_{\mathsf{ideal}}\right] \cdot \Pr\left[\text{E}_{\mathsf{ideal}}\right] \\ &\quad + \Pr\left[\text{OUT}_{\mathsf{ideal}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{ideal}}\right] \cdot \Pr\left[\neg P_2^{\perp} \mid \text{E}_{\mathsf{ideal}}\right] \cdot \Pr\left[\text{E}_{\mathsf{ideal}}\right].\end{aligned}$$

Note that trivially we have

$$\Pr\left[\neg P_2^{\perp} \mid \text{E}_{\mathsf{hyb}}\right] = \Pr\left[\neg P_2^{\perp} \mid \text{E}_{\mathsf{ideal}}\right]$$

and

$$\Pr\left[P_2^{\perp} \mid \text{E}_{\mathsf{hyb}}\right] = \Pr\left[P_2^{\perp} \mid \text{E}_{\mathsf{ideal}}\right].$$

Furthermore, by the definition of the protocol, if P_2 aborts, P_3 outputs $b_{\mathsf{out}} = x_3$ (just as in the previous claim). It is easy to see that this is true in the ideal world as well, so we have

$$\Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right] = \Pr\left[\text{OUT}_{\mathsf{ideal}} = b_{\mathsf{out}} \mid P_2^{\perp} \bigwedge \text{E}_{\mathsf{ideal}}\right].$$

When P_2 does not abort, in both worlds $b_{\mathsf{out}} = b_1^{(i-1)}$. So as long as we can prove that

$$\Pr\left[\text{E}_{\mathsf{hyb}}\right] = \Pr\left[\text{E}_{\mathsf{ideal}}\right] \tag{4.4}$$

it will then follow that

$$\Pr\left[\text{OUT}_{\mathsf{ideal}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{ideal}}\right] = \Pr\left[\text{OUT}_{\mathsf{hyb}} = b_{\mathsf{out}} \mid \neg P_2^{\perp} \bigwedge \text{E}_{\mathsf{hyb}}\right]$$

which will complete the proof of our claim. Before proceeding, we make one final simplification of Equation 4.4. Recall that any view $\vec{a}_i$ of $\mathcal{A}$ (after the completion of ShareGen) consists simply of the values $b_3^{(1)}, \ldots, b_3^{(i)}, b_1^{(i-1)}$. Letting $\text{VIEW}_{\mathsf{hyb}}^{i-1}$ (respectively $\text{VIEW}_{\mathsf{ideal}}^{i-1}$) denote the values received by $\mathcal{A}$ in the first $i-1$ rounds of the protocol in the hybrid (resp. ideal) world, and $\text{VIEW}_{\mathsf{hyb}}^{i}$ (resp. $\text{VIEW}_{\mathsf{ideal}}^{i}$) denote the round i message, along with the following final message received by $\mathcal{A}$ after it aborts P_1 in round i, we note that:

$$\begin{aligned}
&\Pr\left[\text{E}_{\mathsf{hyb}}\right] \\
&\quad = \Pr\left[\text{VIEW}_{\mathsf{hyb}}^{i} = (b_3^{(i)}, b_1^{(i-1)}) \mid \text{VIEW}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1} \bigwedge i \leq i^*\right] \cdot \Pr\left[\text{VIEW}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1} \bigwedge i \leq i^*\right]
\end{aligned}$$

and, equivalently in the ideal world:

$$\begin{aligned}
&\Pr\left[\text{E}_{\mathsf{ideal}}\right] \\
&\quad = \Pr\left[\text{VIEW}_{\mathsf{ideal}}^{i} = (b_3^{(i)}, b_1^{(i-1)}) \mid \text{VIEW}_{\mathsf{ideal}}^{i-1} = \vec{a}_{i-1} \bigwedge i \leq i^*\right] \cdot \Pr\left[\text{VIEW}_{\mathsf{ideal}}^{i-1} = \vec{a}_{i-1} \bigwedge i \leq i^*\right]
\end{aligned}$$

It is trivially true from the protocol and simulator descriptions that

$$\Pr\left[\text{VIEW}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1} \bigwedge i \leq i^*\right] = \Pr\left[\text{VIEW}_{\mathsf{ideal}}^{i-1} = \vec{a}_{i-1} \bigwedge i \leq i^*\right]$$

Furthermore, conditioned $i \leq i^*$, we know that $\text{VIEW}_{\mathsf{hyb}}^{i}$ (resp., $\text{VIEW}_{\mathsf{ideal}}^{i}$) is independent of $\text{VIEW}_{\mathsf{hyb}}^{i-1}$ (resp., $\text{VIEW}_{\mathsf{ideal}}^{i-1}$). Therefore, to prove Equation 4.4, and thus Theorem 4.1.1, it suffices to prove that

$$\Pr\left[\text{VIEW}_{\mathsf{hyb}}^{i} = (b_3^{(i)}, b_1^{(i-1)}) \mid i \leq i^*\right] = \Pr\left[\text{VIEW}_{\mathsf{ideal}}^{i} = (b_3^{(i)}, b_1^{(i-1)}) \mid i \leq i^*\right]$$

We proceed now to do this by looking at every possible set of inputs (x_1, x_2, x_3).

Case 1: if $(\mathbf{x_1 = x_2 = x_3})$, then

$$\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, x_1)\right] = \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, x_1)\right] = 1.$$

In both worlds, $b_3^{(i)}$ is always x_1. When P_1 aborts in the ideal world, in accordance with step 7a, $\mathcal{S}$ chooses $\hat{x}_1 = x_1 = x_2$ with probability $\frac{3}{8}$ and sends $b_1^{(i-1)} = x_1$ to $\mathcal{A}$. If $\mathcal{S}$ chooses $\hat{x}_1 \neq x_1$, then it submits $(\hat{x}_1, x_2)$ for $\hat{x}_1 \neq x_2$ to the trusted party, and $b_{\mathsf{out}} = x_3 = x_1$, so again $b_1^{(i-1)} = x_1$. The analysis is even simpler in the hybrid world, as both values are always x_1.

Case 2: if $(\mathbf{x_1 = x_2 \neq x_3})$, then

$$\begin{aligned}
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, x_1)\right] &= \left((1-\alpha)\cdot\frac{3}{8}\right) + \alpha = \frac{1}{2} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, x_1)\right] &= \left((1-\alpha)\cdot\frac{1}{2}\right) + \left(\alpha\cdot\frac{1}{2}\right) = \frac{1}{2} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, \overline{x}_1)\right] &= (1-\alpha)\cdot\frac{5}{8} = \frac{1}{2} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, \overline{x}_1)\right] &= \left((1-\alpha)\cdot\frac{1}{2}\right) + \left(\alpha\cdot\frac{1}{2}\right) = \frac{1}{2}.
\end{aligned}$$

Case 3: if $(\mathbf{x_3 = x_1 \neq x_2})$, then

$$\begin{aligned}
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, x_1)\right] &= \left(\frac{1}{2}(1-\alpha)\cdot\frac{1}{4}\right) + \alpha = \frac{3}{10} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, x_1)\right] &= \left(\frac{1}{2}(1-\alpha)\cdot\frac{1}{2}\right) + \left(\alpha\cdot\frac{1}{2}\right) = \frac{3}{10} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, \overline{x}_1)\right] &= \frac{1}{2}(1-\alpha)\cdot\frac{3}{4} = \frac{3}{10} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, \overline{x}_1)\right] &= \left(\frac{1}{2}(1-\alpha)\cdot\frac{1}{2}\right) + \left(\alpha\cdot\frac{1}{2}\right) = \frac{3}{10} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (\overline{x}_1, x_1)\right] &= \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (\overline{x}_1, x_1)\right] = \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (\overline{x}_1, \overline{x}_1)\right] &= \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (\overline{x}_1, \overline{x}_1)\right] = \frac{1}{2}(1-\alpha)\cdot\frac{1}{2} = \frac{1}{5}.
\end{aligned}$$

Case 4: if $(\mathbf{x_1 \neq x_2 = x_3})$, then

$$\begin{aligned}
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, x_1)\right] &= \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, x_1)\right] = 0 \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (x_1, \overline{x}_1)\right] &= \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (x_1, \overline{x}_1)\right] = \frac{1}{2}(1-\alpha) = \frac{2}{5} \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (\overline{x}_1, x_1)\right] &= \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (\overline{x}_1, x_1)\right] = 0 \\
\Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{ideal}} = (\overline{x}_1, \overline{x}_1)\right] &= \Pr\left[(b_3^{(i)}, b_{i-1}^{(1)})_{\mathsf{hyb}} = (\overline{x}_1, \overline{x}_1)\right] = \frac{1}{2}(1-\alpha) + \alpha = \frac{3}{5}.
\end{aligned}$$

The key observation with this last set of inputs is that when $x_2 = x_3$, and $i < i^*$, regardless of what value $\mathcal{S}$ chooses for $\hat{x}_1$, $b_1^{(i-1)} = x_2 = x_3$, just as in the hybrid world.

■
■
■

4.2 A Lower Bound on the Round Complexity of Majority

In this section we prove that any fair protocol for three-party majority must have a round complexity of $\omega(\log \kappa)$. In addition to providing evidence that the protocol of Section 4.1 is

optimal, this result also suggests that achieving fairness in the multi-party setting is qualitatively harder than achieving it in the two-party setting. More formally, consider a function $f : \{0,1\} \times \cdots \times \{0,1\} \to \{0,1\}$ taking n boolean inputs. For any subset $\emptyset \subset I \subset [n]$, we can define the *partition* f_I of f to be the two-input function $f_I : \{0,1\}^{|I|} \times \{0,1\}^{n-|I|}$ defined as

$$f_I(y, z) = f(x),$$

where $x \in \{0,1\}^n$ is such that $x_I = y$ and $x_{\bar{I}} = z$. It is not hard to see that if there exists an I for which f_I *cannot* be computed with complete fairness in the two-party setting, then f cannot be computed with complete fairness in the multi-party setting. Similarly, the round complexity for computing f with complete fairness in the multi-party case must be at least the round complexity of fairly computing each f_I. But what about the converse? Note that any partition of the majority function (maj) is just (isomorphic to) the greater-than-or-equal-to function, where the domain of one input can be viewed as $\{0,1,2\}$ and the domain of the other input can be viewed as $\{0,1\}$ (in each case, representing the number of '1' inputs held). We have already demonstrated in Chapter 3 that under suitable cryptographic assumptions, the greater-than-or-equal-to function on constant-size domains can be securely computed with complete fairness in $\mathcal{O}(1)$ rounds. Since the majority function cannot be computed in constant rounds, it seems our protocol from Section 3.1 can not be easily extended to the multi-party setting through a "partition-based" approach.

4.2.1 Proof Overview

We prove our lower bound by arguing that if Π_{maj} is some protocol for securely computing maj, then eliminating the last round of Π_{maj} results in a protocol Π'_{maj} that still computes maj correctly "with high probability". Specifically, if the error probability in Π_{maj} is at most μ (that we will eventually set to some negligible function of κ), then the error probability in Π'_{maj} is at most $c \cdot \mu$ for some constant c. If the original protocol Π_{maj} has $m = \mathcal{O}(\log \kappa)$ rounds, then applying this argument inductively m times gives a protocol that computes maj correctly on all inputs with probability significantly better than guessing *without any interaction at all*. This gives the desired contradiction.

To prove that eliminating the last round of Π_{maj} cannot affect correctness "too much", we consider a constraint that holds for the ideal-world evaluation of maj. (Recall, we are working in the ideal world where complete fairness holds.) Consider an adversary who corrupts two parties, and let the input of the honest party P be chosen uniformly at random. The adversary can learn P's input by submitting $(0,1)$ or $(1,0)$ to the trusted party. The adversary can also try to bias the output of maj to be the opposite of P's choice by submitting $(0,0)$ or $(1,1)$; this will succeed in biasing the result half the time. But the adversary cannot both learn P's input and *simultaneously* bias the result. (If the adversary submits $(0,1)$ or $(1,0)$, the output of maj is always equal to P's input; if the adversary submits $(0,0)$ or $(1,1)$ then the output of maj reveals nothing about P's input.) Concretely, for any ideal-world adversary the *sum* of the probability that the adversary guesses P's input and the probability that the output of maj is not equal to P's input is at most 1. In our proof, we show that if correctness holds with significantly lower probability when the last round of Π_{maj} is eliminated, then there exists a real-world adversary violating this constraint.

4.2.2 Proof Details

We number the parties P_1, P_2, P_3, and work modulo 3 in the subscript. The input of P_j is denoted by x_j. The following claim formalizes the ideal-world constraint described informally above.

Claim 4.2.1 *For all $j \in \{1,2,3\}$ and any adversary $\mathcal{A}$ corrupting P_{j-1} and P_{j+1} in an ideal-world computation of* maj*, we have*

$$\Pr[\mathcal{A} \text{ correctly guesses } x_j] + \Pr[\text{OUT}_j \neq x_j] \leq 1,$$

where the probabilities are taken over the random coins of $\mathcal{A}$ and random choice of $x_j \in \{0,1\}$.

Proof: Consider an execution in the ideal world, where P_j's input x_j is chosen uniformly at random. Let EQUAL be the event that $\mathcal{A}$ submits two equal inputs (i.e., $x_{j-1} = x_{j+1}$) to the trusted party. In this case, $\mathcal{A}$ learns nothing about P_j's input and so can guess x_j with probability at most $1/2$. It follows that:

$$\Pr[\mathcal{A} \text{ correctly guesses } x_j] \leq \frac{1}{2}\Pr[\text{EQUAL}] + \Pr[\overline{\text{EQUAL}}].$$

Moreover, $\Pr[\text{OUT}_j \neq x_j] = \frac{1}{2}\Pr[\text{EQUAL}]$ since $\text{OUT}_j \neq x_j$ occurs only if $\mathcal{A}$ submits $x_{j-1} = x_{j+1} = \bar{x}_j$ to the trusted party. Therefore:

$$\begin{aligned} &\Pr[\mathcal{A} \text{ correctly guesses } x_j] + \Pr[\text{OUT}_j \neq x_j] \\ &\quad\leq \frac{1}{2}\Pr[\text{EQUAL}] + \Pr[\overline{\text{EQUAL}}] + \frac{1}{2}\Pr[\text{EQUAL}] \\ &\quad= \Pr[\text{EQUAL}] + \Pr[\overline{\text{EQUAL}}] = 1, \end{aligned}$$

proving the claim. ■

Let Π_{maj} be a protocol that securely computes maj using $m = m(\kappa)$ rounds. Consider an execution of Π_{maj} in which all parties run the protocol honestly except for possibly aborting in some round. We denote by $b_j^{(i)}$ the value that P_{j-1} and P_{j+1} both[2] output if P_j aborts the protocol after sending its round-i message (and then P_{j-1} and P_{j+1} honestly run the protocol to completion). Similarly, we denote by $b_{j-1}^{(i)}$ (resp., $b_{j+1}^{(i)}$) the value output by P_j and P_{j+1} (resp., P_j and P_{j-1}) when P_{j-1} (resp., P_{j+1}) aborts after sending its round-i message. Note that an adversary who corrupts, e.g., both P_{j-1} and P_{j+1} can compute $b_j^{(i)}$ immediately after receiving the round-i message of P_j.

Since Π_{maj} securely computes maj with complete fairness, the ideal-world constraint from the previous claim implies that for all $j \in \{1,2,3\}$, any inverse polynomial $\mu(\kappa)$, and any poly-time adversary $\mathcal{A}$ controlling players P_{j-1} and P_{j+1}, we have:

$$\Pr_{x_j \leftarrow \{0,1\}}[\mathcal{A} \text{ correctly guesses } x_j] + \Pr_{x_j \leftarrow \{0,1\}}[\text{OUT}_j \neq x_j] \leq 1 + \mu(\kappa) \tag{4.5}$$

for κ sufficiently large. Security of Π_{maj} also guarantees that if the inputs of the honest parties agree, then with all but negligible probability their output must be their common input

[2]Security of Π_{maj} implies that the outputs of P_{j-1} and P_{j+1} in this case must be equal with all but negligible probability. For simplicity we assume this to hold with probability 1 but our proof can be modified easily to remove this assumption.

regardless of when a malicious P_j aborts. That is, for κ large enough we have

$$x_{j+1} = x_{j-1} \;\Rightarrow\; \Pr\left[b_j^{(i)} = x_{j+1} = x_{j-1}\right] \geq 1 - \mu(\kappa) \tag{4.6}$$

for all $j \in \{1,2,3\}$ and all $i \in \{0,\ldots,m(\kappa)\}$.

The following claim represents the key step in our lower bound.

Claim 4.2.2 *Fix a protocol Π_{maj}, a function μ, and a value κ such that Equations (4.5) and (4.6) hold, and let $\mu = \mu(\kappa)$. Say there exists an i, with $1 \leq i \leq m(\kappa)$, such that for all $j \in \{1,2,3\}$ and all $c_1, c_2, c_3 \in \{0,1\}$ it holds that:*

$$\Pr\left[b_j^{(i)} = \mathsf{maj}(c_1,c_2,c_3) \mid (x_1,x_2,x_3) = (c_1,c_2,c_3)\right] \geq 1 - \mu. \tag{4.7}$$

Then for all $j \in \{1,2,3\}$ and all $c_1, c_2, c_3 \in \{0,1\}$ it holds that:

$$\Pr\left[b_j^{(i-1)} = \mathsf{maj}(c_1,c_2,c_3) \mid (x_1,x_2,x_3) = (c_1,c_2,c_3)\right] \geq 1 - 5\mu. \tag{4.8}$$

Proof: When $j = 1$ and $c_2 = c_3$, the desired result follows from Equation (4.6); this is similarly true for $j = 2$, $c_1 = c_3$ as well as $j = 3$, $c_1 = c_2$.

Consider the real-world adversary $\mathcal{A}$ that corrupts P_1 and P_3 and sets $x_1 = 0$ and $x_3 = 1$. Then:

- $\mathcal{A}$ runs the protocol honestly until it receives the round-i message from P_2.
- $\mathcal{A}$ then locally computes the value of $b_2^{(i)}$.
  - If $b_2^{(i)} = 0$, then $\mathcal{A}$ aborts P_1 *without sending its round-i message* and runs the protocol (honestly) on behalf of P_3 until the end. By definition, the output of P_2 will be $b_1^{(i-1)}$.
  - If $b_2^{(i)} = 1$, then $\mathcal{A}$ aborts P_3 *without sending its round-i message* and runs the protocol (honestly) on behalf of P_1 until the end. By definition, the output of P_2 will be $b_3^{(i-1)}$.
- After completion of the protocol, $\mathcal{A}$ outputs $b_2^{(i)}$ as its guess for the input of P_2.

Consider an experiment in which the input x_2 of P_2 is chosen uniformly at random, and then $\mathcal{A}$ runs protocol Π_{maj} with P_2. Using Equation (4.7), we have:

$$\begin{aligned} \Pr\left[\mathcal{A} \text{ correctly guesses } x_2\right] &= \Pr\left[b_2^{(i)} = x_2\right] \\ &= \Pr\left[b_2^{(i)} = f(0,x_2,1)\right] \;\geq\; 1 - \mu\,. \end{aligned} \tag{4.9}$$

We also have:

$$\begin{aligned}\Pr\left[\text{OUT}_2 \neq x_2\right] &= \frac{1}{2}\cdot\Pr\left[\text{OUT}_2 = 1 \mid (x_1,x_2,x_3) = (0,0,1)\right] \\ &\quad + \frac{1}{2}\cdot\Pr\left[\text{OUT}_2 = 0 \mid (x_1,x_2,x_3) = (0,1,1)\right] \\ &= \frac{1}{2}\Big(\Pr\left[b_1^{(i-1)} = 1 \wedge b_2^{(i)} = 0 \mid (x_1,x_2,x_3) = (0,0,1)\right] \\ &\quad + \Pr\left[b_3^{(i-1)} = 1 \wedge b_2^{(i)} = 1 \mid (x_1,x_2,x_3) = (0,0,1)\right] \\ &\quad + \Pr\left[b_3^{(i-1)} = 0 \wedge b_2^{(i)} = 1 \mid (x_1,x_2,x_3) = (0,1,1)\right] \\ &\quad + \Pr\left[b_1^{(i-1)} = 0 \wedge b_2^{(i)} = 0 \mid (x_1,x_2,x_3) = (0,1,1)\right]\Big)\,.\end{aligned} \tag{4.10}$$

From Equation (4.5), we know that the sum of Equations (4.9) and (4.10) is upper-bounded by $1+\mu$. Looking at the first summand in Equation (4.10), this implies that

$$\Pr\left[b_1^{(i-1)} = 1 \wedge b_2^{(i)} = 0 \mid (x_1,x_2,x_3) = (0,0,1)\right] \leq 4\mu. \tag{4.11}$$

Probabilistic manipulation gives

$$\begin{aligned}&\Pr\left[b_1^{(i-1)} = 1 \wedge b_2^{(i)} = 0 \mid (x_1,x_2,x_3) = (0,0,1)\right] \\ &= 1 - \Pr\left[b_1^{(i-1)} = 0 \vee b_2^{(i)} = 1 \mid (x_1,x_2,x_3) = (0,0,1)\right] \\ &\geq 1 - \Pr\left[b_1^{(i-1)} = 0 \mid (x_1,x_2,x_3) = (0,0,1)\right] - \Pr\left[b_2^{(i)} = 1 \mid (x_1,x_2,x_3) = (0,0,1)\right] \\ &\geq 1 - \Pr\left[b_1^{(i-1)} = 0 \mid (x_1,x_2,x_3) = (0,0,1)\right] - \mu\,,\end{aligned}$$

where the last inequality is due to the assumption of the claim. Combined with Equation (4.11), this implies:

$$\Pr\left[b_1^{(i-1)} = 0 \mid (x_1,x_2,x_3) = (0,0,1)\right] \geq 1 - 5\mu.$$

Applying an analogous argument starting with the third summand in Equation (4.10) gives

$$\Pr\left[b_3^{(i-1)} = 1 \mid (x_1,x_2,x_3) = (0,1,1)\right] \geq 1 - 5\mu.$$

Repeating the entire argument, but modifying the adversary to consider all possible pairs of corrupted parties and all possible settings of their inputs, completes the proof of the claim. ■

Theorem 4.2.3 *Any protocol Π_{maj} that securely computes* maj *with complete fairness requires $\omega(\log\kappa)$ rounds.*

Proof: Assume there exists a protocol Π_{maj} that securely computes maj with complete fairness using $m = \mathcal{O}(\log\kappa)$ rounds. Let $\mu(\kappa) = \frac{1}{4\cdot 5^{m(\kappa)}}$, and note that μ is noticeable. By the assumed security of Π_{maj}, the conditions of Claim 4.2.2 hold for κ large enough; Equation (4.7), in particular, holds for $i = m(\kappa)$. Fixing this κ and applying the claim iteratively $m(\kappa)$ times, we conclude that P_{j-1} and P_{j+1} can correctly compute the value of the function, on all inputs, with probability at least $3/4$ *without interacting with P_j at all*. This is clearly impossible. ■

Π_{OR}

Inputs: Each party P_i holds input $x_i \in \{0,1\}$, and the security parameter is k.

Computation:

1. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of all players.
2. Each player P_i chooses random coins r_i and broadcasts $c_i = \mathsf{Com}(1^\kappa, x_i, r_i)$, where Com denotes a computationally-hiding, statistically-binding commitment scheme. If any party P_i does not broadcast anything (or otherwise broadcasts an invalid value), then all honest players output 1. Otherwise, let $\mathbf{c} = (c_1, \ldots, c_n)$.
3. All players $P_i \in \mathcal{P}$ run a protocol $\pi_{\mathcal{P}}$ for computing $\mathsf{CommittedOR}_{\mathcal{P}}$, with party P_i using $(x_i, r_i, \mathbf{c}_{\mathcal{P}})$ as its input where $\mathbf{c}_{\mathcal{P}} \stackrel{\text{def}}{=} (c_i)_{i:P_i \in \mathcal{P}}$.
4. If players receive $\perp$ from the execution of $\mathsf{CommittedOR}_{\mathcal{P}}$, they set $\mathcal{P} = \mathcal{P} \setminus \{P^*\}$, where $P^* \in \mathcal{P}$ is the lowest-indexed player in $\mathcal{P}$, and return to step 3.
5. If players receive a set $\mathcal{D} \subset \mathcal{P}$ from the execution of $\mathsf{CommittedOR}_{\mathcal{P}}$, they set $\mathcal{P} = \mathcal{P} \setminus \mathcal{D}$ and return to step 3.
6. If players receive a binary output from the execution of $\mathsf{CommittedOR}_{\mathcal{P}}$, they output this value and end the protocol.

Figure 4.3: A protocol computing OR for n players.

4.3 Fair Computation of OR for n Players

In this section, we demonstrate the completely-fair computation of a non-trivial function for an arbitrary number of parties n, any $t < n$ of whom are corrupted. Specifically, we show how to compute boolean OR with complete fairness. The idea in the protocol is to have the parties repeatedly try to compute OR *on committed inputs*, using a protocol that is secure-with-designated-abort (i.e., where only the lowest-indexed party can force an abort. See Definition 2.6.1.) The key observation is that, in case of an abort, the dishonest players only learn something about the inputs of the honest players if all the malicious parties use input 0. (If any of the malicious players holds input 1, then the output is always 1 regardless of the inputs of the honest parties.) So, if the lowest-indexed party is corrupt and aborts the computation of the committed OR, then the remaining parties simply recompute the committed OR using '0' as the effective input for any parties that have already been eliminated. They repeatedly proceed in this fashion, eliminating dishonest parties at each iteration. Eventually, when the lowest-indexed player is honest, the process terminates and all honest players receive (correct) output.

The actual protocol follows the above intuition, but is a bit more involved. A formal description of the protocol is given in Figure 4.3, and the "committed OR" functionality is defined in Figure 4.4.

Theorem 4.3.1 *Assume* Com *is a computationally-hiding, statistically-binding commitment scheme, and that $\pi_{\mathcal{P}}$ securely computes* $\mathsf{CommittedOR}_{\mathcal{P}}$ *(with abort). Then the protocol of Figure 4.3 computes* OR *with complete fairness.*

Proof: Let Π denote the protocol of Figure 4.3. For simplicity we assume Com is perfectly binding, though statistical binding suffices. For any non-uniform, polynomial time adversary $\mathcal{A}$ in the hybrid world, we demonstrate a non-uniform polynomial-time adversary $\mathcal{S}$ corrupting the same parties as $\mathcal{A}$ and running in the ideal world with access to an ideal functionality

$\mathsf{CommittedOR}_{\mathcal{P}}$

Inputs: The functionality is run by parties in $\mathcal{P}$. Let the input of player $P_i \in \mathcal{P}$ be $(x_i, r_i, \mathbf{c}^i)$ where $\mathbf{c}^i = (c^i_j)_{j:P_j \in \mathcal{P}}$. The security parameter is k.

For each party $P_i \in \mathcal{P}$, determine its output as follows:

1. Say P_j *disagrees with* P_i if either (1) $\mathbf{c}^j \neq \mathbf{c}^i$ or (2) $\mathsf{Com}(1^\kappa, x_j, r_j) \neq c^i_j$. (Note that disagreement is not a symmetric relation.)
2. Let $\mathcal{D}_i$ be the set of parties who disagree with P_i.
3. If there exist any parties that disagree with each other, return $\mathcal{D}_i$ as output to P_i. Otherwise, return $\bigvee_{j:P_j \in \mathcal{P}} x_j$ to all parties.

Figure 4.4: Functionality $\mathsf{CommittedOR}_{\mathcal{P}}$, parameterized by a set $\mathcal{P}$

computing OR (with complete fairness), such that

$$\left\{\text{IDEAL}_{\mathsf{OR},\mathcal{S}}(x_1,\ldots,x_n,k)\right\}_{x_i\in\{0,1\},k\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{HYBRID}^{\mathsf{CommittedOR}_{\mathcal{P}}}_{\Pi,\mathcal{A}}(x_1,\ldots,x_n,k)\right\}_{x_i\in\{0,1\},k\in\mathbb{N}}.$$

Applying the composition theorem of [20] then proves the theorem.

We note that when no players are corrupt, the proof of security is easy, due to the assumed existence of a private broadcast channel. We now describe the execution of $\mathcal{S}$:

1. Let $\mathcal{C} \neq \emptyset$ be the corrupted players, and let $\mathcal{H} = \{P_1, \ldots P_n\} \setminus \mathcal{C}$ denote the honest players. Initialize $\mathcal{I} = \mathcal{C}$. Looking ahead, $\mathcal{I}$ denotes the set of corrupted parties who have not yet been eliminated from the protocol.

2. $\mathcal{S}$ invokes $\mathcal{A}$ on the inputs $\{x_i\}_{i:P_i\in\mathcal{C}}$, the auxiliary input z, and the security parameter k.

3. For $P_i \in \mathcal{H}$, the simulator $\mathcal{S}$ gives to $\mathcal{A}$ a commitment $c_i = \mathsf{Com}(1^\kappa, x_i, r_i)$ to $x_i = 0$ using randomness r_i. $\mathcal{S}$ then records the commitment c_i that is broadcast by $\mathcal{A}$ on behalf of each party $P_i \in \mathcal{C}$. If any corrupted player fails to broadcast a value c_i, then $\mathcal{S}$ submits 1's to the trusted party on behalf of all corrupted parties, outputs whatever $\mathcal{A}$ outputs, and halts.

4. If $\mathcal{I} = \emptyset$, $\mathcal{S}$ submits (on behalf of all the corrupted parties) 0's to the trusted party computing OR (unless it has already done so). It then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{I} \neq \emptyset$, continue to the next step.

5. $\mathcal{S}$ sets $\mathcal{P} = \mathcal{H} \cup \mathcal{I}$ and obtains inputs $\{(r_i, x_i, \mathbf{c}^i)\}_{i:P_i\in\mathcal{I}}$ for the computation of $\mathsf{CommittedOR}_{\mathcal{P}}$. For each $P_i \in \mathcal{P}$, the simulator $\mathcal{S}$ computes the list of players $\mathcal{D}_i$ that disagree with P_i (as in Figure 4.4), using as the inputs of the honest parties the commitments defined in Step 3, and assuming that honest parties provide correct decommitments. Observe that if $P_i, P_j \in \mathcal{H}$ then $\mathcal{D}_i = \mathcal{D}_j \subseteq \mathcal{I}$. Let $\mathcal{D}_\mathcal{H} \subseteq \mathcal{I}$ be the set of parties that disagree with the honest parties.

   Let P^* be the lowest-indexed player in $\mathcal{P}$. If no parties disagree with each other, go to step 6. Otherwise:

   (a) If $P^* \in \mathcal{I}$, then $\mathcal{A}$ is given $\{\mathcal{D}_i\}_{i:P_i\in\mathcal{I}}$. If P^* aborts, then $\mathcal{S}$ sets $\mathcal{I} = \mathcal{I} \setminus \{P^*\}$ and goes to step 4. If P^* does not abort, then $\mathcal{S}$ sets $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}_\mathcal{H}$ and goes to step 4.

   (b) If $P^* \notin \mathcal{I}$, then $\mathcal{A}$ is given $\{\mathcal{D}_i\}_{i:P_i\in\mathcal{I}}$. Then $\mathcal{S}$ sets $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}_\mathcal{H}$ and goes to step 4.

6. $\mathcal{S}$ computes the value $b = \bigvee_{P_i \in \mathcal{I}} x_i$.

   (a) If $b = 0$, and $\mathcal{S}$ has not yet queried the trusted party computing OR, then $\mathcal{S}$ submits 0's (on behalf of all the corrupted parties) to the trusted party and stores the output of the trusted party as b_{out}. $\mathcal{S}$ gives b_{out} to $\mathcal{A}$ (either as just received from the trusted party, or as stored in a previous execution of this step).

   (b) If $b = 1$, then $\mathcal{S}$ gives the value 1 to $\mathcal{A}$ *without* querying the trusted party.

   $\mathcal{S}$ now continues as follows:

   (a) If $P^* \in \mathcal{I}$ and P^* aborts, then $\mathcal{S}$ sets $\mathcal{I} = \mathcal{I} \setminus \{P^*\}$ and goes to step 4.

   (b) If $P^* \notin \mathcal{I}$, or if P^* does not abort, then $\mathcal{S}$ submits 1's to the trusted party if it has not yet submitted 0's. It outputs whatever $\mathcal{A}$ outputs, and halts.

We first demonstrate that the view of $\mathcal{A}$ in the hybrid world is computationally indistinguishable from its view in the ideal world. In step 3, the commitments by the simulator are all commitments to 0 values rather than to the actual inputs of the honest players. However, it is easy to see that if $\mathcal{A}$ could distinguish between the two worlds in this step, he could violate the security of the underlying commitment scheme. We now show that, except for Step 3 of the simulation, the ideal world view generated by the simulator and the hybrid world view are identically distributed. Recall that at the start of Step 5 of the simulation, we let $\mathcal{P} = \mathcal{I} \cup \mathcal{H}$ denote the set of remaining players. We first note that when the outputs of $\mathsf{CommittedOR}_{\mathcal{P}}(r_i, x_i, \mathbf{c}^i)_{i:P_i \in \mathcal{I} \cup \mathcal{H}}$ are disagreement lists (rather than the OR of the remaining inputs), then in Step 5, $\mathcal{S}$ is capable both of correctly detecting this, and of computing the disagreement lists, independently of the honest input values. In this step, then, the view of $\mathcal{A}$ will be identical to his view in the hybrid world. If all remaining players are consistent, in which case the output of $\mathsf{CommittedOR}_{\mathcal{P}}$ is the binary OR of the remaining inputs, there are two possibilities. If the input x_i for some $P_i \in \mathcal{I}$ is 1, then the adversarial view created by $\mathcal{S}$ in Step 6b is exactly as in the hybrid world; the output of $\mathsf{CommittedOR}_{\mathcal{I} \cup \mathcal{H}}$ in the hybrid world is always 1 in this case, regardless of the honest players' inputs. When all input values x_i for $P_i \in \mathcal{I}$ are 0, then the hybrid world output of $\mathsf{CommittedOR}_{\mathcal{I} \cup \mathcal{H}}$ will depend on the inputs of the honest players, and $\mathcal{S}$ must query the trusted party to determine this value. Note, however, that in this case the output of $\mathsf{CommittedOR}_{\mathcal{I} \cup \mathcal{H}}$ in all subsequent calls in the hybrid world will remain unchanged, regardless of which players are later excluded, and thus the view generated by $\mathcal{S}$ in Step 6a is correct every time.

We next consider the joint distribution of the honest players' outputs with the view of $\mathcal{A}$. We claim that the output of the honest players (in both worlds) is exactly $b = \bigvee_{P_i} x_i$ for P_i that are never eliminated, where for honest parties, these are simply their original input values, and for malicious parties these are the values they first committed to (either in Step 3 in the ideal world, or in Step 2 in the hybrid world). In the hybrid world, this claim follows trivially from the protocol description. In the ideal world, there are two possible submissions that $\mathcal{S}$ can make to the trusted party: $\mathcal{S}$ can submit all 0's or all 1's. $\mathcal{S}$ submits 0's to the trusted party when all (remaining) inputs x_i for $P_i \in \mathcal{I}$ are 0 (in Step 6a), or when $\mathcal{I} = \emptyset$ (in Step 4). In this case the output of the honest parties is

$$b = \bigvee_{P_i \in \mathcal{I} \cup \mathcal{H}} x_i = \bigvee_{P_i \in \mathcal{I}' \cup \mathcal{H}} x_i$$

for any $\mathcal{I}' \subseteq \mathcal{I}$. Therefore, regardless of which players from $\mathcal{I}$ are eliminated in the future, the output of the honest parties is equal to the OR of the inputs of the non-eliminated players, as

claimed. The only time $\mathcal{S}$ submits 1's to the trusted party is in Step 6b, after it has verified that no more parties will abort, and that (at least) one of the remaining inputs is a 1. Here too, then, the output of the honest players is consistent with the inputs of the non-eliminated players. Finally, notice that the set of players still participating at the end of the protocol depends only on the view of $\mathcal{A}$. Since we have already argued that the distributions on $\mathcal{A}$'s views in the two worlds are computationally indistinguishable, it follows that the distribution on possible sets of non-eliminated players, $\mathcal{I} \cup \mathcal{H}$ are computationally indistinguishable as well. This completes the proof sketch. ■

Chapter 5

Partial Fairness in Secure Two-Party Computation

In Chapter 1 we discussed extensively the various approaches that have been suggested for achieving partial fairness. For the most part, they have come in two (similar) flavors: those that make it easier in each round to compute the output through brute-force, and those that have given the players increasing statistical confidence in the output in each round. In this Chapter we introduce our own definition of partial fairness, given below, and we explore the feasibility of achieving this definition. This work originally appeared in Eurocrypt 2010 [52].

As discussed, the most desirable (but, in the two-party setting, typically unachievable) definition of security requires computational indistinguishability between the real world and a "true" ideal world where parties receive output simultaneously. The usual relaxation of security-with-abort [42] leaves unchanged the requirement of computational indistinguishability, but weakens the ideal world so that fairness is no longer guaranteed at all. Motivated by [58], we suggest an alternate relaxation: keep the ideal world unchanged, *but relax the notion of simulation* and require instead that the real and ideal worlds be distinguishable with probability at most $\frac{1}{p} + \mathsf{negl}$, where p is some specified polynomial[1] (see Definition 5.0.2). We refer to a protocol satisfying this definition as being "$\frac{1}{p}$-secure". Cleve [25] and Moran et al. [74] show $\frac{1}{p}$-secure protocols for two-party coin tossing (where parties have no inputs), but we are not aware of any other results satisfying our definition. In particular, none of the prior approaches for achieving partial fairness yield protocols that are $\frac{1}{p}$-secure.

We propose the notion of $\frac{1}{p}$-security as a new way to approach the problem of partial fairness, and view this as an independent contribution. We also demonstrate protocols that achieve this definition for a broad class of functionalities. Specifically, let $f_\kappa : X_\kappa \times Y_\kappa \to Z^1_\kappa \times Z^2_\kappa$ be a (randomized) functionality where player 1 (resp., player 2) provides input $x \in X_\kappa$ (resp., $y \in Y_\kappa$) and receives output $z^1 \in Z^1_\kappa$ (resp., $z^2 \in Z^2_\kappa$). For arbitrary polynomial p, we show $\frac{1}{p}$-secure protocols for computing f_κ as long as at least one of $X_\kappa, Y_\kappa, Z^1_\kappa, Z^2_\kappa$ is polynomial size (in κ). Our protocols are always *private*, and when either X_κ or Y_κ is polynomial-size we also achieve the usual notion of security-with-abort. We assume only the existence of en-

[1] Katz [58] proposed a related notion of "$\frac{1}{p}$-security-with-abort"; that definition, however, continues to compare the real world with the relaxed ideal world and so again guarantees no fairness at all. A similar relaxation, formalized differently and with different motivation (and again giving no fairness), is also used in [6]. Our definition of $\frac{1}{p}$-security is also similar in spirit to (but weaker than) the notion of ϵ-zero knowledge [31] and is analogous to some definitions of password-based key exchange [44] (although there p is fixed by the size of the password dictionary).

hanced trapdoor permutations or, more generally, oblivious transfer.

We also prove that our feasibility results are, in general, *optimal*. First, we demonstrate a deterministic, boolean function $f_\kappa : X_\kappa \times Y_\kappa \to \{0,1\}$, where X_κ and Y_κ both have super-polynomial size, for which no protocol computing f_κ can simultaneously achieve both security-with-abort and $\frac{1}{p}$-security (for $p > 4$). We also show a deterministic function $f_\kappa : X_\kappa \times Y_\kappa \to Z_\kappa$, with each of $X_\kappa, Y_\kappa, Z_\kappa$ super-polynomial in size, such that f_κ cannot be $\frac{1}{p}$-securely computed for $p > 2$.

Preliminaries: For a fixed function p, the ensembles $X = \{X(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\mathcal{D}_\kappa,\,\kappa\in\mathbb{N}}$ are *computationally $\frac{1}{p}$-indistinguishable*, denoted $X \overset{1/p}{\approx} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every κ and every $a \in \mathcal{D}_\kappa$

$$\left|\Pr[D(X(a,\kappa)) = 1] - \Pr[D(Y(a,\kappa)) = 1]\right| \leq \frac{1}{p(\kappa)} + \mu(\kappa).$$

This gives the following definition:

Definition 5.0.2 *Protocol π is said to $\frac{1}{p}$-*securely compute *$\mathcal{F}$ if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_\kappa\times Y_\kappa,\, z\in\{0,1\}^*,\,\kappa\in\mathbb{N}} \overset{1/p}{\approx} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x,y,\kappa)\right\}_{(x,y)\in X_\kappa\times Y_\kappa,\, z\in\{0,1\}^*,\,\kappa\in\mathbb{N}}.$$

5.1 $\frac{1}{\mathbf{p}}$-Secure Computation of General Functionalities

We begin with an informal description of our feasibility results. Let x denote the input of P_1, let y denote the input of P_2, and let $f : X \times Y \to Z$ denote the function they are trying to compute. (For simplicity, here we omit the dependence of X, Y, and Z on κ, and focus on the case where each party receives the same output.) As in previous chapters, our protocols will be composed of two stages, where in the first stage the players execute a ShareGen protocol, and in the second stage they exchange shares of outputs in a sequence of $m = m(\kappa)$ iterations. Here, instead of choosing i^* from a geometric distribution, as we did in previous chapters, we choose i^* uniformly from $\{1,\ldots,m\}$. When X and Y (the domains of f) have size polynomial in κ, the second stage follows that of Π_{EXOR} from Chapter 3: we set $a_i = f(x,\hat{y})$ for $\hat{y}$ chosen uniformly from Y, and set $b_i = f(\hat{x},y)$ for $\hat{x}$ chosen uniformly (and independently) from X. Note that a_i (resp., b_i) is independent of y (resp., x), as desired.

Intuitively, this is partially fair because fairness is violated only if P_1 aborts *exactly* in iteration i^*. (If P_1 aborts before iteration i^* then neither party learns the "correct" value $z = f(x,y)$, while if he aborts after iteration i^* then both parties learn the correct value. An abort by P_2 in iteration i^* does not violate fairness, since by then P_1 has already learned the output.) We show that *even if P_1 knows the value of z* (which it may, depending on partial information P_1 has about y), it cannot determine with certainty when iteration i^* occurs. Specifically, we prove a general result (see Lemma 5.1.1) implying (roughly) that as long as $\Pr[a_i = z] \geq \alpha$ for all $i < i^*$, then P_1 cannot abort in iteration i^* except with probability at most $1/\alpha m$ (recall that m is the number of iterations in the second phase). Since $\Pr[a_i = f(x,y)] = \Pr_{\hat{y}\in Y}[f(x,\hat{y}) = f(x,y)] \geq \Pr_{\hat{y}\in Y}[\hat{y} = y] = 1/|Y|$ for any x, y, we conclude that setting $m = p \cdot |Y|$, so that

$1/\alpha m = 1/p$, suffices to achieve $\frac{1}{p}$-security. We thus get a protocol with polynomially many rounds as long as Y is polynomial size.

The above does not work directly when Y has super-polynomial size. To fix this, we must ensure that for every possible $z \in Z$ (the range of f) we have that $\Pr[a_i = z]$ is noticeable. We do this by changing the distribution of a_i (for $i < i^*$) as follows: with probability $1 - 1/q$ choose a_i as above, but with probability $1/q$ choose a_i uniformly from Z. Now, for any x, y, we have $\Pr[a_i = f(x,y)] \geq \frac{1}{q} \cdot \Pr_{a_i \in Z}[a_i = f(x,y)] \geq 1/q|Z|$ and so setting $m = pq|Z|$ ensures that P_1 cannot abort in iteration i^* except with probability at most $1/p$.

Changing the distribution of a_i, however, introduces a new problem: if P_2 aborts prior to iteration i^*, the output of the honest P_1 in the real world cannot necessarily be simulated in the ideal world. We show, however, that it *can* be simulated to within statistical difference $\mathcal{O}(1/q)$. Taking $q = p$ (along with $m = pq|Z|$) thus gives a $\frac{1}{p}$-secure protocol with polynomially many rounds.

We begin in Section 5.1.1 by stating a lemma that forms an essential piece of our analysis in the two sections that follow. In Section 5.1.2 we demonstrate a private and $\frac{1}{p}$-secure protocol for functionalities defined on polynomial-size domains. A slight modification of this protocol is also simultaneously secure-with-abort. To keep the exposition as simple as possible, we restrict our attention there to single-output functionalities (though the techniques extend easily to the general case). In Section 5.1.3 we show how to adapt the protocol for functionalities defined over domains of super-polynomial size (but polynomial range), and also generalize to functionalities generating different outputs for each party.

5.1.1 A Useful Lemma

We analyze an abstract game Γ between a challenger and an (unbounded) adversary $\mathcal{A}$. The game is parameterized by a value $\alpha \in (0, 1]$ and an integer $m \geq 1$. Fix arbitrary distributions D_1, D_2 such that for every z it holds that

$$\Pr_{a \leftarrow D_1}[a = z] \geq \alpha \cdot \Pr_{a \leftarrow D_2}[a = z]. \tag{5.1}$$

The game $\Gamma(\alpha, m)$ proceeds as follows:

1. The challenger chooses i^* uniformly from $\{1, \ldots, m\}$, and then chooses $a_1, \ldots, a_m$ as follows:
   - For $i < i^*$, it chooses $a_i \leftarrow D_1$.
   - For $i \geq i^*$, it chooses $a_i \leftarrow D_2$.
2. The challenger and $\mathcal{A}$ then interact in a sequence of at most m iterations. In iteration i:
   - The challenger gives a_i to the adversary.
   - The adversary can either abort or continue. In the former case, the game stops. In the latter case, the game continues to the next iteration.
3. $\mathcal{A}$ *wins* if it aborts the game in iteration i^*.

Let $\mathsf{Win}(\alpha, m)$ denote the maximum probability with which $\mathcal{A}$ wins the above game.

Lemma 5.1.1 *For any D_1, D_2 satisfying Equation (5.1), it holds that $\mathsf{Win}(\alpha, m) \leq 1/\alpha m$.*

Proof: Fix D_1, D_2 satisfying (5.1). We prove the lemma by induction on m. When $m = 1$ the lemma is trivially true; for completeness, we also directly analyze the case $m = 2$. Since $\mathcal{A}$ is unbounded we may assume it is deterministic. So without loss of generality, we may assume the adversary's strategy is determined by a set S in the support of D_2 such that $\mathcal{A}$ aborts in the first iteration iff $a_1 \in S$, and otherwise aborts in the second iteration (no matter what). We have

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins and } i^* = 1] + \Pr[\mathcal{A} \text{ wins and } i^* = 2] \\
&= \frac{1}{2} \cdot \Pr_{a \leftarrow D_2}[a \in S] + \frac{1}{2} \cdot \left(1 - \Pr_{a \leftarrow D_1}[a \in S]\right) \\
&\leq \frac{1}{2} \cdot \Pr_{a \leftarrow D_2}[a \in S] + \frac{1}{2} \cdot \left(1 - \alpha \cdot \Pr_{a \leftarrow D_2}[a \in S]\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left((1-\alpha) \cdot \Pr_{a \leftarrow D_2}[a \in S]\right) \;\leq\; 1 - \alpha/2,
\end{aligned}$$

where the first inequality is due to Equation (5.1). One can easily verify that $1 - \alpha/2 \leq 1/2\alpha$ when $\alpha > 0$. We have thus proved $\mathsf{Win}(\alpha, 2) \leq 1/2\alpha$.

Assume $\mathsf{Win}(\alpha, m) \leq 1/\alpha m$, and we now bound $\mathsf{Win}(\alpha, m+1)$. As above, let S denote a set in the support of D_2 such that $\mathcal{A}$ aborts in the first iteration iff $a_1 \in S$. If $\mathcal{A}$ does *not* abort in the first iteration, and the game does not end, then the conditional distribution of i^* is uniform in $\{2, \ldots, m+1\}$ and the game $\Gamma(\alpha, m+1)$ from this point forward is exactly equivalent to the game $\Gamma(\alpha, m)$. In particular, conditioned on the game $\Gamma(\alpha, m+1)$ not ending after the first iteration, the best strategy for $\mathcal{A}$ is to play whatever is the best strategy in game $\Gamma(\alpha, m)$. We thus have

$$\begin{aligned}
\mathsf{Win}(\alpha, m+1) &= \Pr[\mathcal{A} \text{ wins and } i^* = 1] + \Pr[\mathcal{A} \text{ wins and } i^* > 1] \\
&= \frac{1}{m+1} \cdot \Pr_{a \leftarrow D_2}[a \in S] + \frac{r}{m+1} \cdot \left(1 - \Pr_{a \leftarrow D_1}[a \in S]\right) \cdot \mathsf{Win}(\alpha, m) \\
&\leq \frac{1}{m+1} \cdot \Pr_{a \leftarrow D_2}[a \in S] + \frac{1}{\alpha(m+1)} \cdot \left(1 - \alpha \cdot \Pr_{a \leftarrow D_2}[a \in S]\right) \cdot \\
&= \frac{1}{\alpha(m+1)}\,.
\end{aligned}$$

This completes the proof. ∎

5.1.2 $\frac{1}{\mathrm{p}}$-Security for Functionalities with Polynomial-Size Domain

In this section, we describe a protocol that works for functionalities where at least one of the domains is polynomial-size. (We stress that the protocol works *directly* for randomized functionalities; the standard reduction from randomized to deterministic functionalities [42] would not apply here since, in general, it makes the domain too large.) Although a small modification of the protocol works even when the parties receive different outputs, for simplicity we assume here that the parties compute a single-output function. We return to the more general setting in the following section.

Theorem 5.1.2 *Let $\mathcal{F} = \{f_\kappa : X_\kappa \times Y_\kappa \to Z_\kappa\}$ be a (randomized) functionality where $|Y_\kappa| = \mathsf{poly}(\kappa)$. Assuming the existence of enhanced trapdoor permutations, for any polynomial p there is an $\mathcal{O}(p \cdot |Y_\kappa|)$-round protocol computing $\mathcal{F}$ that is private and $\frac{1}{p}$-secure.*

Proof: As described earlier, our protocol Π consists of two stages. Let p be a polynomial, and set $m = p(\kappa) \cdot |Y_\kappa|$. We implement the first stage of Π using a sub-protocol π for com-

$\mathsf{ShareGen}_m$

Inputs: Let the inputs to $\mathsf{ShareGen}_m$ be $x \in X_\kappa$ and $y \in Y_\kappa$. (If one of the received inputs is not in the correct domain, a default input is substituted.)

Computation:

1. Define values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$ in the following way:
   - Choose i^* uniformly at random from $\{1, \ldots, m\}$.
   - For $i = 1$ to $i^* - 1$ do:
     - Choose $\hat{y} \leftarrow Y_\kappa$ and set $a_i = f_\kappa(x, \hat{y})$.
     - Choose $\hat{x} \leftarrow X_\kappa$ and set $b_i = f_\kappa(\hat{x}, y)$.
   - Compute $z = f_\kappa(x, y)$. For $i = i^*$ to m, set $a_i = b_i = z$.
2. For $1 \leq i \leq m$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (I.e., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)
3. Compute $k_a, k_b \leftarrow \mathsf{Gen}(1^\kappa)$. For $1 \leq i \leq m$, let $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ and $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$.

Output:

1. Send to P_1 the values $a_1^{(1)}, \ldots, a_m^{(1)}$ and $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and the MAC-key k_a.
2. Send to P_2 the values $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$ and $b_1^{(2)}, \ldots, b_m^{(2)}$, and the MAC-key k_b.

Figure 5.1: Functionality $\mathsf{ShareGen}_m$.

puting a randomized functionality $\mathsf{ShareGen}_m$ (parameterized by a polynomial m) that is defined in Figure 5.1. This functionality returns shares to each party, authenticated using an information-theoretically secure m-time MAC $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$. In the second stage of Π the parties exchange these shares in a sequence of m iterations as described in Figure 5.2.

We analyze our protocol in a hybrid model where there is a trusted party computing $\mathsf{ShareGen}_m$ according to the ideal model described in Definition 2.4.1, where a malicious P_1 can abort the trusted party before it sends output to the honest party. We prove privacy and $\frac{1}{p}$-security of Π in this hybrid model; it follows as in [20] that if we use a sub-protocol for computing $\mathsf{ShareGen}_m$ that is secure-with-abort, then the real-world protocol Π is private and $\frac{1}{p}$-secure.

We first consider the case of a malicious P_1. Intuition for the following claim was given in Section 5.1. The formal statement and proof follow.

Claim 5.1.3 *Let Π^{hy} denote an execution of Π in a hybrid model with access to an ideal functionality computing $\mathsf{ShareGen}_m$ (with abort). For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_1 and running Π^{hy}, there exists a non-uniform, polynomial-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing $\mathcal{F}$ (with complete fairness), such that $\frac{1}{p}$-security holds, i.e.,*

$$\left\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*} \overset{1/p}{\approx} \left\{\mathrm{HYBRID}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*},$$

and privacy holds, i.e.,

$$\left\{\mathrm{OUT}^{\mathcal{S}}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*} \equiv \left\{\mathrm{VIEW}^{\mathcal{A}}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*}.$$

Proof: We construct a simulator $\mathcal{S}$ that is given black-box access to $\mathcal{A}$. *For readability in what follows, we ignore the MAC-tags and keys.* That is, we do not mention the fact that $\mathcal{S}$ computes

Protocol 1

Inputs: Party P_1 has input x and party P_2 has input y. Let $m = p \cdot |Y_\kappa|$.

The protocol:

1. **Preliminary phase:**
   (a) P_1 chooses $\hat{y} \in Y_\kappa$ uniformly at random, and sets $a_0 = f_\kappa(x, \hat{y})$. Similarly, P_2 chooses $\hat{x} \in X_\kappa$ uniformly at random, and sets $b_0 = f_\kappa(\hat{x}, y)$.
   (b) Parties P_1 and P_2 run a protocol π to compute $\mathsf{ShareGen}_m$, using their inputs x and y.
   (c) If P_2 receives $\perp$ from the above computation, it outputs b_0 and halts. Otherwise, the parties proceed to the next step.
   (d) Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and k_a.
   (e) Denote the output of P_2 from π by $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, $b_1^{(2)}, \ldots, b_m^{(2)}$, and k_b.
2. **For $i = 1, \ldots, r$ do:**
   P_2 sends the next share to P_1:
   (a) P_2 sends $(a_i^{(2)}, t_i^a)$ to P_1.
   (b) P_1 receives $(a_i^{(2)}, t_i^a)$ from P_2. If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 0$ (or if P_1 received an invalid message, or no message), then P_1 outputs a_{i-1} and halts.
   (c) If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 1$, then P_1 sets $a_i = a_i^{(1)} \oplus a_i^{(2)}$ (and continues running the protocol).
   P_1 sends the next share to P_2:
   (a) P_1 sends $(b_i^{(1)}, t_i^b)$ to P_2.
   (b) P_2 receives $(b_i^{(1)}, t_i^b)$ from P_1. If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 0$ (or if P_2 received an invalid message, or no message), then P_2 outputs b_{i-1} and halts.
   (c) If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 1$, then P_2 sets $b_i = b_i^{(1)} \oplus b_i^{(2)}$ (and continues running the protocol).
3. If all m iterations have been run, party P_1 outputs a_m and party P_2 outputs b_m.

Figure 5.2: Generic protocol for computing a functionality f_κ.

MAC-tags for messages it sends to $\mathcal{A}$, nor do we mention the fact that $\mathcal{S}$ must verify the MAC-tags on the messages sent by $\mathcal{A}$. When we say that $\mathcal{A}$ "aborts", we include in this the event that $\mathcal{A}$ sends an invalid message, or a message whose tag does not pass verification. We also drop the subscript n from our notation and write X, Y in place of X_κ, Y_κ.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input[2] x', the auxiliary input, and the security parameter n. The simulator also chooses $\hat{x} \in X$ uniformly at random (it will send $\hat{x}$ to the trusted party, if needed).
2. $\mathcal{S}$ receives the input x of $\mathcal{A}$ to the computation of the functionality $\mathsf{ShareGen}_m$. (If $x \notin X$ a default input is substituted.)
3. $\mathcal{S}$ sets $m = p \cdot |Y|$, and chooses uniformly-distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. Then, $\mathcal{S}$ gives these shares to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}_m$.

[2] We reserve x for the value input by $\mathcal{A}$ to the computation of $\mathsf{ShareGen}_m$.

4. If $\mathcal{A}$ sends abort to the trusted party computing $\mathsf{ShareGen}_m$, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.

5. Choose i^* uniformly from $\{1, \ldots, m\}$

6. For $i = 1$ to $i^* - 1$:

   (a) $\mathcal{S}$ chooses $\hat{y} \in Y$ uniformly at random, computes $a_i = f(x, \hat{y})$, and sets $a_i^{(2)} = a_i^{(1)} \oplus a_i$. It gives $a_i^{(2)}$ to $\mathcal{A}$. (A fresh $\hat{y}$ is chosen in every iteration.)

   (b) If $\mathcal{A}$ aborts, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party, outputs whatever $\mathcal{A}$ outputs, and halts.

7. For $i = i^*$ to m:

   (a) If $i = i^*$ then $\mathcal{S}$ sends x to the trusted party computing f and receives $z = f(x, y)$.

   (b) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus z$ and gives $a_i^{(2)}$ to $\mathcal{A}$.

   (c) If $\mathcal{A}$ aborts, then $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

8. If $\mathcal{A}$ never aborted (and all m iterations are done), $\mathcal{S}$ outputs what $\mathcal{A}$ outputs and halts.

It is immediate that the view of $\mathcal{A}$ in the simulation above is distributed identically to its view in Π^{hy}; privacy follows. We now prove $\frac{1}{p}$-security.

Ignoring the possibility of a MAC forgery, we claim that the statistical difference between an execution of $\mathcal{A}$, running Π in a hybrid world with access to an ideal functionality computing $\mathsf{ShareGen}_m$, and an execution of $\mathcal{S}$, running in an ideal world with access to an ideal functionality computing f, is at most $1/p$. (Thus, taking into account the possibility of a MAC forgery makes the statistical difference at most $1/p + \mu(\kappa)$ for some negligible function μ.) To see this, let y denote the input of the honest P_2 and consider three cases depending on when the adversary aborts:

1. $\mathcal{A}$ aborts in round $i < i^*$. Conditioned on this event, the view of $\mathcal{A}$ is identically distributed in the two worlds (and is independent of y), and the output of the honest party is $f(\hat{x}, y)$ for $\hat{x}$ chosen uniformly in X.

2. $\mathcal{A}$ aborts in round $i > i^*$ (or never). Conditioned on this, the view of $\mathcal{A}$ is again distributed identically in the two worlds, and in both worlds the output of the honest party is $f(x, y)$.

3. $\mathcal{A}$ aborts in round $i = i^*$: here, although the view of $\mathcal{A}$ is still identical in both worlds, the output of the honest party is not: in the hybrid world the honest party will output $f(\hat{x}, y)$, for $\hat{x}$ chosen uniformly in X, while in the ideal world the honest party will output $f(x, y)$.

   However, Lemma 5.1.1 implies that this event occurs with probability at most $1/p$. To see this, let D_1 denote the distribution of a_i for $i < i^*$ (i.e., this is the distribution defined by the output of $f(x, \hat{y})$, for $\hat{y}$ chosen uniformly from Y), and let D_2 denote the distribution of a_{i^*} (i.e., the distribution defined by the output of $f(x, y)$). For any $z \in Z$ we

have

$$\begin{aligned}\Pr_{a\leftarrow D_1}[a=z] &\stackrel{\text{def}}{=} \Pr_{\hat{y}\leftarrow Y}[f(x,\hat{y})=z] \\ &\geq \frac{1}{|Y|}\cdot\Pr[f(x,y)=z] \;=\; \frac{1}{|Y|}\cdot\Pr_{a\leftarrow D_2}[a=z].\end{aligned}$$

Taking $\alpha = 1/|Y|$ and applying Lemma 5.1.1, we see that $\mathcal{A}$ aborts in iteration i^* with probability at most $1/\alpha m = |Y|/|Y|p = 1/p$.

This completes the proof of the claim. ∎

Next we consider the case of a malicious P_2. A proof of the following is almost identical to that of Claim 5.1.3; in fact, the proof is simpler and we can prove a stronger notion of security (namely, where complete fairness holds) since P_1 always "gets the output first" in every iteration of Π. For these reasons, a proof is omitted.

Claim 5.1.4 *Let Π^{hy} denote an execution of Π in a hybrid model with access to an ideal functionality computing $\mathsf{ShareGen}_m$ (with abort). For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_2 and running Π^{hy}, there exists a non-uniform, polynomial-time adversary $\mathcal{S}$ corrupting P_2 and running in the ideal world with access to an ideal functionality computing $\mathcal{F}$ (with complete fairness), such that $\frac{1}{p}$-security holds, i.e.,*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x,y,\kappa)\right\}_{x\in X_\kappa, y\in Y_\kappa, \mathsf{aux}\in\{0,1\}^*} \overset{1/p}{\approx} \left\{\text{HYBRID}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x,y,\kappa)\right\}_{x\in X_\kappa, y\in Y_\kappa, \mathsf{aux}\in\{0,1\}^*},$$

and privacy holds, i.e.,

$$\left\{\text{OUT}^{\mathcal{S}}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x,y,\kappa)\right\}_{x\in X_\kappa, y\in Y_\kappa, \mathsf{aux}\in\{0,1\}^*} \equiv \left\{\text{VIEW}^{\mathcal{A}}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x,y,\kappa)\right\}_{x\in X_\kappa, y\in Y_\kappa, \mathsf{aux}\in\{0,1\}^*}.$$

The results of [20], along with the fact that a secure-with-abort protocol for $\mathsf{ShareGen}_m$ is implied by the existence of enhanced trapdoor permutations, complete the proof of Theorem 5.1.2. ∎

Achieving security-with-abort. As written, the protocol is not secure-with-abort. However, the protocol can be modified easily so that it is (without affecting $\frac{1}{p}$-security): simply have $\mathsf{ShareGen}_m$ choose i^* uniformly from $\{2,\ldots,m+1\}$ and set $b_{i^*-1} = \perp$, where $\perp$ is some distinguished value outside the range of f. Although this allows a malicious P_2 to identify exactly when iteration i^* occurs, this does not affect security since by that time P_1 has already received the correct output.

5.1.3 $\frac{1}{\mathrm{p}}$-Security for Functionalities with Polynomial-Size Range

The protocol from the previous section does not apply to functions on domains of super-polynomial size, since the round complexity is linear in the size of the smaller domain. Here we show how to extend the protocol to handle arbitrary domains if the range of the function (for at least one of the parties) is polynomial size. We now also explicitly take into account the case when parties obtain different outputs. Intuition for the changes we introduce is given in Section 5.1.

Theorem 5.1.5 *Let $\mathcal{F} = \{f_\kappa : X_\kappa \times Y_\kappa \to Z^1_\kappa \times Z^2_\kappa\}$ be a (randomized) functionality, where $|Z^1_\kappa| = \mathsf{poly}(\kappa)$. Assuming the existence of enhanced trapdoor permutations, for any polynomial p there is an $\mathcal{O}\left(p^2\cdot|Z^1_\kappa|\right)$-round protocol computing $\mathcal{F}$ that is private and $\frac{1}{p}$-secure.*

$\mathsf{ShareGen}_{p,m}$

Inputs: Let the inputs to $\mathsf{ShareGen}_{p,m}$ be $x \in X_\kappa$ and $y \in Y_\kappa$. (If one of the received inputs is not in the correct domain, a default input is substituted.)

Computation:

1. Define values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$ in the following way:
   - Choose i^* uniformly at random from $\{1, \ldots, m\}$.
   - For $i = 1$ to $i^* - 1$ do:
     - Choose $\hat{x} \leftarrow X_\kappa$ and set $b_i = f_\kappa^2(\hat{x}, y)$.
     - With probability $\frac{1}{p}$, choose $z \leftarrow Z_\kappa^1$ and set $a_i = z$. With the remaining probability $1 - \frac{1}{p}$, choose $\hat{y} \leftarrow Y$ and set $a_i = f_\kappa^1(x, \hat{y})$.
   - Compute $z_1 = f_\kappa^1(x, y)$ and $z_2 = f_\kappa^2(x, y)$ (if $f_\kappa = (f_\kappa^1, f_\kappa^2)$ is randomized, these values are computed using the same random tape). For $i = i^*$ to m, set $a_i = z_1$ and $b_i = z_2$.
2. For $1 \leq i \leq m$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (E.g., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)
3. Compute $k_a, k_b \leftarrow \mathsf{Gen}(1^\kappa)$. For $1 \leq i \leq m$, let $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ and $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$.

Output:

1. Send to P_1 the values $a_1^{(1)}, \ldots, a_m^{(1)}$ and $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and the MAC-key k_a.
2. Send to P_2 the values $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$ and $b_1^{(2)}, \ldots, b_m^{(2)}$, and the MAC-key k_b.

Figure 5.3: Functionality $\mathsf{ShareGen}_{p,m}$.

Proof: Our protocol Π is, once again, composed of two stages. The second stage is identical to the second stage of the previous protocol (see Figure 5.2), except that the number of iterations m is now set to $m = p^2 \cdot |Z_\kappa^1|$. The first stage generates shares using a sub-routine π computing a different functionality $\mathsf{ShareGen}_{p,m}$, parameterized by both p and m and described in Figure 5.3.

We again analyze our protocol in a hybrid model, where there is now a trusted party computing $\mathsf{ShareGen}_{p,m}$. (Once again, P_1 can abort the computation of $\mathsf{ShareGen}_{p,m}$ in the ideal world.) We prove privacy and $\frac{1}{p}$-security of Π in this hybrid model, implying [20] that if the parties use a secure-with-abort protocol for computing $\mathsf{ShareGen}_{p,m}$, then the real-world protocol Π is private and $\frac{1}{p}$-secure. We first consider the case of a malicious P_1.

Claim 5.1.6 *Let Π^{hy} denote an execution of Π in a hybrid model with access to an ideal functionality computing $\mathsf{ShareGen}_{p,m}$ (with abort). For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_1 and running Π^{hy}, there exists a non-uniform, polynomial-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing $\mathcal{F}$ (with complete fairness), such that $\frac{1}{p}$-security holds, i.e.,*

$$\left\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*} \overset{1/p}{\approx} \left\{\mathrm{HYBRID}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*},$$

and privacy holds, i.e.,

$$\left\{\mathrm{OUT}^{\mathcal{S}}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*} \equiv \left\{\mathrm{VIEW}^{\mathcal{A}}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x, y, \kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*}.$$

Proof: The simulator used to prove this claim is essentially the same as the simulator used in the proof of Claim 5.1.3, except that in step 6(a) the distribution on a_i (for $i < i^*$) is changed to the one used by $\mathsf{ShareGen}_{p,m}$. The analysis is similar, too, except for bounding the probability that $\mathcal{A}$ aborts in iteration i^*. To bound this probability we will again rely on Lemma 5.1.1, but now distribution D_1 (i.e., the distribution of a_i for $i < i^*$) is different. Let y denote the input of P_2. Note that, by construction of $\mathsf{ShareGen}_{p,m}$, for any $z \in Z_\kappa^1$ we have $\Pr_{a \leftarrow D_1}[a = z] \geq \frac{1}{p} \cdot \frac{1}{|Z_\kappa^1|}$. Regardless of f^1 and y, it therefore holds for all $z \in Z_\kappa^1$ that

$$\Pr_{a \leftarrow D_1}[a = z] \geq \frac{1}{p \cdot |Z_\kappa^1|} \cdot \Pr_{a \leftarrow D_2}[a = z].$$

Setting $\alpha = 1/p \cdot |Z_\kappa^1|$ and applying Lemma 5.1.1, we see that $\mathcal{A}$ aborts in iteration i^* with probability at most

$$\frac{1}{\alpha m} = \frac{p \cdot |Z_\kappa^1|}{p^2 \cdot |Z_\kappa^1|} = \frac{1}{p}.$$

This completes the proof of the claim. ■

We next consider the case of a malicious P_2. Note that, in contrast to Claim 5.1.4, here we claim only $\frac{1}{p}$-security (and privacy).

Claim 5.1.7 *Let Π^{hy} denote an execution of Π in a hybrid model with access to an ideal functionality computing $\mathsf{ShareGen}_{p,m}$ (with abort). For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_2 and running Π^{hy}, there exists a non-uniform, polynomial-time adversary $\mathcal{S}$ corrupting P_2 and running in the ideal world with access to an ideal functionality computing $\mathcal{F}$ (with complete fairness), such that $\frac{1}{p}$-security holds, i.e.,*

$$\left\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x,y,\kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*} \overset{1/p}{\approx} \left\{\mathrm{HYBRID}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x,y,\kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*},$$

and privacy holds, i.e.,

$$\left\{\mathrm{OUT}^{\mathcal{S}}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x,y,\kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*} \equiv \left\{\mathrm{VIEW}^{\mathcal{A}}_{\Pi^{\mathsf{hy}},\mathcal{A}(\mathsf{aux})}(x,y,\kappa)\right\}_{x \in X_\kappa, y \in Y_\kappa, \mathsf{aux} \in \{0,1\}^*}.$$

Proof: The simulator $\mathcal{S}$ in this case is fairly obvious, but we include it for completeness. Once again, for readability we ignore the presence of the MAC-tags and keys.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input y', the auxiliary input, and the security parameter n. The simulator also chooses $\hat{y} \in Y$ uniformly at random (it will send $\hat{y}$ to the trusted party, if needed).

2. $\mathcal{S}$ receives the input y of $\mathcal{A}$ to the computation of the functionality $\mathsf{ShareGen}_{p,m}$. (If $y \notin Y$ a default input is substituted.)

3. $\mathcal{S}$ sets $m = p^2 \cdot |Z^1|$, and chooses uniformly-distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. Then, $\mathcal{S}$ gives these shares to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}_{p,m}$.

4. Choose i^* uniformly from $\{1, \ldots, m\}$

5. For $i = 1$ to $i^* - 1$:

(a) $\mathcal{S}$ chooses $\hat{x} \in X$ uniformly at random, computes $b_i = f^2(\hat{x}, y)$, and sets $b_i^{(1)} = b_i^{(2)} \oplus b_i$. It gives $b_i^{(1)}$ to $\mathcal{A}$. (Note that a fresh $\hat{x}$ is chosen in every iteration.)

(b) If $\mathcal{A}$ aborts, then $\mathcal{S}$ sends $\hat{y}$ to the trusted party, outputs whatever $\mathcal{A}$ outputs, and halts.

6. For $i = i^*$ to m:

   (a) If $i = i^*$ then $\mathcal{S}$ sends y to the trusted party computing f and receives $z = f^2(x, y)$.

   (b) $\mathcal{S}$ sets $b_i^{(1)} = b_i^{(2)} \oplus z$ and gives $b_i^{(1)}$ to $\mathcal{A}$.

   (c) If $\mathcal{A}$ aborts, then $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

7. If $\mathcal{A}$ has never aborted (and all m iterations are done), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

Privacy is immediate, and so we focus on $\frac{1}{p}$-security. Ignoring the possibility of a MAC forgery, we claim that the statistical difference between an execution of $\mathcal{A}$, running Π in a hybrid world with access to an ideal functionality computing $\mathsf{ShareGen}_{p,m}$, and an execution of $\mathcal{S}$, running in an ideal world with access to an ideal functionality computing $\mathcal{F}$, is at most $1/p$. (Thus, taking into account the possibility of a MAC forgery makes the statistical difference at most $1/p + \mu(\kappa)$ for some negligible function μ.) The view of $\mathcal{A}$ is identical in the two worlds; the only issue is the output of the honest P_1 holding input x. Specifically, if $\mathcal{A}$ aborts in any iteration prior to i^* then, in the ideal-world interaction with $\mathcal{S}$, party P_1 outputs $f^1(x, \hat{y})$ for a uniformly-chosen $\hat{y} \in Y$. In the hybrid world, however, the output of P_1 is given by the distribution of a_i (for $i < i^*$) as determined by $\mathsf{ShareGen}_{p,m}$. However, these two distributions are within statistical difference (at most) $1/p$. The claim follows. ■

The results of [20], along with the fact that a secure-with-abort protocol for $\mathsf{ShareGen}_{p,m}$ is implied by the existence of enhanced trapdoor permutations, complete the proof of Theorem 5.1.5. ■

5.2 Optimality of Our Results

We show that the results of the previous section are optimal as far as generic feasibility is concerned.

5.2.1 Impossibility of $\frac{1}{p}$-Security and Security-with-Abort Simultaneously

In Section 5.1.2 (cf. the remark at the end of that section) we showed a protocol achieving $\frac{1}{p}$-security and security-with-abort *simultaneously* for functionalities where at least one of the domains is polynomial-size. We show that if both domains are super-polynomial in size then, in general, it is impossible to achieve both these criteria at once.

Theorem 5.2.1 *Let $\mathcal{F} = \left\{\mathsf{EQ}_\kappa : \{0,1\}^{\ell(\kappa)} \times \{0,1\}^{\ell(\kappa)} \to \{0,1\}\right\}$, where EQ_κ denotes the equality function on strings and $\ell(\kappa) = \omega(\log \kappa)$. Let Π be any protocol computing $\mathcal{F}$. If Π is secure-with-abort, then Π does not $\frac{1}{p}$-securely compute $\mathcal{F}$ for any $p \geq 4 + \frac{1}{\mathsf{poly}(\kappa)}$.*

Proof: Let Π be a protocol that computes $\mathcal{F}$ and is secure-with-abort. Assume without loss of generality that P_2 sends the first message in Π and that P_1 sends the last message. Say Π has $m = m(\kappa)$ iterations for some polynomial m, where an iteration consists of a message sent by P_2 followed by a message sent by P_1. Let a_0 denote the value that P_1 outputs if P_2 sends nothing, and let a_i, for $1 \leq i \leq m$, denote the value that P_1 outputs if P_2 aborts after sending its iteration-i message. Similarly, let b_0 denote the value that P_2 outputs if P_1 sends nothing, and let b_i, for $1 \leq i \leq m$, denote the value that P_2 outputs if P_1 aborts after sending its iteration-i message. We may assume without loss of generality that, for all i, we have $a_i \in \{0,1\}$ and $b_i \in \{0,1,\perp\}$.

We will consider two experiments involving an execution of Π. In the first, x and y are chosen uniformly and independently from $\{0,1\}^{\ell(\kappa)}$; the parties are given inputs x and y, respectively; and the parties then run protocol Π honestly. We denote the probability of events in this experiment by $\Pr_{\text{rand}}[\cdot]$. In the second experiment, x is chosen uniformly from $\{0,1\}^{\ell(\kappa)}$ and y is set equal to x; these inputs are given to the parties and they run the protocol honestly as before. We denote the probability of events in this probability space by $\Pr_{\text{eq}}[\cdot]$.

Claim 5.2.2 $\Pr_{\text{rand}}[a_0 = 1 \vee \cdots \vee a_m = 1]$ *and* $\Pr_{\text{rand}}[b_0 = 1 \vee \cdots \vee b_m = 1]$ *are negligible.*

Proof: This follows from the fact that Π is secure-with-abort. If, say, it were the case that $\Pr_{\text{rand}}[a_0 = 1 \vee \cdots \vee a_m = 1]$ is not negligible, then we could consider an adversarial P_2 that runs the protocol honestly but aborts at a random round. This would cause the honest P_1 to output 1 with non-negligible probability in the real world, whereas P_1 outputs 1 with only negligible probability in the ideal world (since the parties are given independent, random inputs). ■

Assume for simplicity that Π has perfect correctness, i.e., that $a_m = b_m = \mathsf{EQ}(x,y)$ when the two parties run the protocol honestly holding initial inputs x and y. (This assumption is not necessary, but allows us to avoid having to deal with annoying technicalities.) Then

$$\Pr_{\text{eq}}[a_0 = 1 \vee \cdots \vee a_m = 1] = \Pr_{\text{eq}}[b_0 = 1 \vee \cdots \vee b_m = 1] = 1$$

since, in particular, $\Pr_{\text{eq}}[a_m = 1] = \Pr_{\text{eq}}[b_m = 1] = 1$. In a given execution, let i^* denote the lowest index for which $a_{i^*} = 1$, and let j^* denote the lowest index for which $b_{j^*} = 1$. Since

$$\Pr_{\text{eq}}[i^* \leq j^*] + \Pr_{\text{eq}}[i^* > j^*] = 1,$$

at least one of the terms on the left-hand side is at least $1/2$. We assume that $\Pr_{\text{eq}}[i^* \leq j^*] \geq 1/2$, but the same argument (swapping the roles of the parties) applies if $\Pr_{\text{eq}}[i^* > j^*] \geq 1/2$.

Consider now a third experiment that is a mixture of the previous two. Specifically, in this experiment a random bit b is chosen; if $b = 0$ then the parties are given inputs x and y as in the first experiment (i.e., chosen uniformly and independently at random), while if $b = 1$ then the parties are given (random) $x = y$ as in the second experiment. The parties then run protocol Π honestly. We denote the probability of events in this probability space by $\Pr_3^{\mathsf{real}}[\cdot]$. We use the superscript real to distinguish this from an ideal-world version of this experiment where the bit b is chosen uniformly and the parties are given x and y generated accordingly, but now the parties interact with an ideal party computing EQ *without abort* (i.e., in the first ideal model). We denote the probability of events in this experiment by $\Pr_3^{\mathsf{ideal}}[\cdot]$.

Consider an execution of the third experiment (in either the real or ideal worlds), in the case when P_1 is malicious. Let guess denote the event that P_1 correctly guesses the value of

the bit b, and let out_2 denote the output of P_2. It is not hard to show that

$$\Pr_3^{\mathsf{ideal}}[\mathsf{guess} \wedge \mathsf{out}_2 \neq 1] = \frac{1}{2}. \tag{5.2}$$

(Note that $\mathsf{out}_2 \in \{0,1\}$ in the first ideal world.) Now take the following real-world adversary $\mathcal{A}$ corrupting P_1: upon receiving input x, adversary $\mathcal{A}$ runs Π honestly but computes a_i after receiving each iteration-i message from P_2. Then:

- If, at some point, $a_i = 1$ then $\mathcal{A}$ aborts the protocol (before sending the iteration-i message on behalf of P_1) and outputs the guess "$b = 1$".
- If $a_i = 0$ for all i, then $\mathcal{A}$ simply runs the protocol to the end (including the final message of the protocol) and outputs the guess "$b = 0$".

We have:

$$\begin{aligned}
&\Pr_3^{\mathsf{real}}[\mathsf{guess} \wedge \mathsf{out}_2 \neq 1] \\
&= \frac{1}{2} \cdot \Pr_{\mathsf{rand}}[\mathsf{guess} \wedge \mathsf{out}_2 \neq 1] + \frac{1}{2} \cdot \Pr_{\mathsf{eq}}[\mathsf{guess} \wedge \mathsf{out}_2 \neq 1] \\
&\geq \frac{1}{2} \cdot \Pr_{\mathsf{rand}}[a_1 = 0 \wedge \cdots \wedge a_m = 0 \wedge b_m = 0] + \frac{1}{2} \cdot \Pr_{\mathsf{eq}}[i^* \leq j^*] \\
&\geq \frac{1}{2} \cdot (1 - \mathsf{negl}(\kappa)) + \frac{1}{4} \;=\; \frac{3}{4} - \mathsf{negl}(\kappa),
\end{aligned} \tag{5.3}$$

using Claim 5.2.2 for the second inequality. Equations (5.2) and (5.3) show that Π cannot also be $\frac{1}{p}$-secure for any $p \geq 4 + \frac{1}{\mathsf{poly}(\kappa)}$. ■

5.2.2 Impossibility of $\frac{1}{p}$-Security for General Functions

Our results show that $\frac{1}{p}$-security is achievable for any functionality $f : X_\kappa \times Y_\kappa \to Z_\kappa^1 \times Z_\kappa^2$ if at least one of $X_\kappa, Y_\kappa, Z_\kappa^1, Z_\kappa^2$ has polynomial size. Here, we demonstrate that this limitation is inherent.

Define a deterministic, single-output function $\mathcal{F} = \{\mathsf{Swap}_\kappa\}$ with

$$\mathsf{Swap}_\kappa : \{0,1\}^{\omega(\log\kappa)} \times \{0,1\}^{\omega(\log\kappa)} \to \{0,1\}^{\omega(\log\kappa)}$$

as follows: Fix some $\ell(\kappa) = \omega(\log\kappa)$. Let $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ denote an information-theoretic, one-time MAC for messages of length $2 \cdot \ell(\kappa)$ with key length $\mathcal{O}(\ell(\kappa))$ and tag length $\ell(\kappa)$. Then

$$\mathsf{Swap}_\kappa\big((x_1, t_1, k_2), (x_2, t_2, k_1)\big) \stackrel{\text{def}}{=} \begin{cases} (x_1, x_2) & \text{if } \mathsf{Vrfy}_{k_1}(x_1, t_1) = \mathsf{Vrfy}_{k_2}(x_2, t_2) = 1 \\ \perp & \text{otherwise} \end{cases}.$$

(Note that both parties receive the same output (x_1, x_2) in the first case.)

Theorem 5.2.3 *Function $\mathcal{F}$ cannot be $\frac{1}{p}$-securely computed for any $p \geq 2 + \frac{1}{\mathsf{poly}(\kappa)}$.*

Proof: Consider an ideal-world computation of Swap where:

- x_1, x_2 are chosen uniformly at random from $\{0,1\}^{2\ell(\kappa)}$.

- k_1, k_1', k_2, k_2' are output by $\mathsf{Gen}(1^\kappa)$ (i.e., they are random MAC-keys).
- $t_1 = \mathsf{Mac}_{k_1}(x_1)$, $t_1' = \mathsf{Mac}_{k_1'}(x_1)$, $t_2 = \mathsf{Mac}_{k_2}(x_2)$, and $t_2' = \mathsf{Mac}_{k_2'}(x_2)$.
- P_1 is given input (x_1, t_1, k_2) and auxiliary information (k_2', t_2')
- P_2 is given input (x_2, t_2, k_1) and auxiliary information (k_1', t_1').

Define a *win* for P_1 as the event that P_1 outputs x_2 while P_2 fails to output x_1. (A win for P_2 is defined analogously.) It is easy to see that, e.g., a malicious P_1 cannot win in the ideal world, where complete fairness is guaranteed, except with negligible probability. This is because x_2 is a uniform $2\ell(\kappa)$-bit value, while the only information P_1 has about x_2 initially is the $\ell(\kappa)$-bit tag t_2'. Thus, the only way for P_1 to learn x_2 is to submit to the trusted party some input $(\hat{x}_1, \hat{t}_1, \hat{k}_2)$ for which $\mathsf{Vrfy}_{k_1}(\hat{x}_1, \hat{t}_1) = 1$; unless $\hat{x}_1 = x_1$, however, this condition holds with negligible probability.

In any real-world computation of Swap, however, there must be one party who "gets its output first" with probability at least $1/2$, and can identify exactly when this occurs using its auxiliary information. More formally, say we have an m-iteration protocol Π computing Swap where P_2 sends the first message and P_1 sends the last message. Let a_i, for $i = 0, \ldots, m$, denote the second component of the value P_1 would output if P_2 aborts the protocol after sending its iteration-i message, and let b_i denote the first component of the value that P_2 would output if P_1 aborts the protocol after sending its iteration-i message. Each value a_i and b_i can be computed in polynomial time after receiving the other party's iteration-i message. We can therefore define an adversary P_1^* that acts as follows:

> Run the protocol honestly until the first round where $\mathsf{Vrfy}_{k_2'}(a_i, t_2') = 1$; then output a_i and abort.

An adversary P_2^* can be defined analogously. Note that if, e.g., $\mathsf{Vrfy}_{k_2'}(a_i, t_2') = 1$ then $a_i = x_2$ except with negligible probability; this follows from the information-theoretic security of the MAC along with the fact that the execution of Π is independent of k_2', t_2'.

Let i denote the first round in which $\mathsf{Vrfy}_{k_2'}(a_i, t_2') = 1$, and let j denote the first round in which $\mathsf{Vrfy}_{k_1'}(b_j, t_1') = 1$. Assuming for simplicity that Π has perfect correctness, we have

$$\Pr[i \leq j] + \Pr[j > i] = 1.$$

Further, since $\big|\Pr[P_1^* \text{ wins}] - \Pr[i \leq j]\big|$ and $\big|\Pr[P_2^* \text{ wins}] - \Pr[i > j]\big|$ are both negligible, we see that either P_1^* or P_2^* wins in the real world with probability at least $1/2 - \mathsf{negl}(\kappa)$. Since an adversary wins in the ideal world with negligible probability, this rules out $\frac{1}{p}$-security for $p > 2$. ∎

Chapter 6

Fairness When Players are Rational

Because we cannot generally achieve complete fairness, our work in Chapter 5 introduced a new way of relaxing the definition of fairness. In this chapter we instead consider relaxing our assumptions about the adversary. Most research in cryptography considers arbitrary malicious behavior, and with good reason: for most applications, and under reasonable assumptions, we can achieve security even under such conditions. Since this is not the case with respect to fair computation, it is reasonable to consider weaker adversaries in this setting. Motivated by the work of Halpern and Teague [53] we consider here a particular application where we desire fair protocols, and we demonstrate feasibility under the assumption that all players act *rationally*. That is, in the traditional game theoretic sense, we assume the players all behave in a way that maximizes their own utility. Of course, if we allowed for arbitrary utility functions, we would again be facing adversaries with arbitrary behavior. Instead, we restrict the utility functions, with the aim of still achieving some meaningful notion of security, while also enabling fairness.

The problem we consider (as Halpern and Teague before us [53]) is called rational secret sharing. The classical problem of t-out-of-n secret sharing [84, 14] (c.f. Section 2.2.3) involves a "dealer" D who wishes to entrust a secret s to a group of n players $P_1, \ldots, P_n$ so that (1) any group of t or more players can reconstruct the secret without further intervention of the dealer, yet (2) any group of fewer than t players has no information about the secret. As an example, consider the scheme due to Shamir [84]: assume the secret s lies in a finite field $\mathcal{F}$, with $|\mathcal{F}| > n$. The dealer chooses a random polynomial $f(x)$ of degree at most $t-1$ subject to the constraint $f(0) = s$, and gives the "share" $f(i)$ to player P_i (for $i = 1, \ldots, n$). Any set of t players can recover $f(x)$ (and hence s) by broadcasting their shares and interpolating the polynomial; furthermore, no set of fewer than t players can deduce any information about s.

A secret sharing scheme, such as Shamir's above, involves two distinct protocols: $\mathsf{Share}(t, n, s)$, which is run by the dealer to generate the shares of s, and $\mathsf{Rec}(s_1, \ldots, s_n)$, which is run by some subset of the players in order to reconstruct s. The goal in *rational* secret sharing is to find a fair reconstruction protocol. In other words, we wish to make sure that while running $\mathsf{Rec}(s_1, \ldots, s_n)$, no player receives s before any other player. As mentioned before, we assume the players are acting to maximize particular utility functions. Specifically, we assume each player prefers first and foremost any outcome where they recover the secret. Then, that being equal, they prefer an outcome where the fewest other players recover the secret. Formally, let $\mu_i(o)$ denote the utility of player P_i for the outcome o. For a particular outcome o of the protocol, we let $\delta_i(o)$ be a bit denoting whether or not P_i learns the secret, and let $\mathsf{num}(o) = \sum_i \delta_i(o)$; i.e., $\mathsf{num}(o)$ is simply the number of players who learn the secret.

Following [53], we make the following assumptions about the utility functions of the players:

- $\delta_i(o) > \delta_i(o') \Rightarrow \mu_i(o) > \mu_i(o')$.
- If $\delta_i(o) = \delta_i(o')$, then $\mathsf{num}(o) < \mathsf{num}(o') \Rightarrow \mu_i(o) > \mu_i(o')$.

When fairness is not a concern, the only necessary interaction required in the reconstruction protocol is to have each player broadcast their share to all other players. However, with the above utility functions, no player has any incentive to do this! Consider P_1: if strictly fewer than $t-1$ other players broadcast their shares to the rest of the group, then no one learns the secret regardless of whether P_1 reveals his share or not. If more than $t-1$ players reveal their shares, then everyone learns the secret and P_1's actions again have no effect. On the other hand, if *exactly* $t-1$ other players reveal their shares, then P_1 learns the secret (using his share), and he can prevent other players from learning the secret by *not* publicly revealing his share.

Let t, n be as above, and let $t^* \geq t$ denote the number of players present when the secret is to be reconstructed. Given the above discussion, we can conclude the following about the game-theoretic equilibria of "standard" Shamir secret sharing in the above situation (definitions of Nash equilibria and weakly dominating strategies are given in Section 6.1):

- For any t, n, t^*, it is a Nash equilibrium for no one to reveal their share.
- If $t^* > t$, it is a Nash equilibrium for all t^* participating players to reveal their shares. However, as discussed above, it is a weakly dominating strategy for each player *not* to reveal his share; thus, the Nash equilibrium likely to be reached is the one mentioned earlier in which no one reveals their share.
- If $t^* = t$, then having all t^* participating players reveal their shares is not even a Nash equilibrium, since each player can profitably deviate by not revealing his share.

Related work: Halpern and Teague [53] introduced the question of rational secret sharing. They proved that no protocol that has a fixed number of rounds can have a Nash equilibrium that results in reconstruction. They demonstrate a probabilistic protocol for $t, n \geq 3$, and claim impossibility for $n = 2$. In contrast, our protocol extends even to $t, n \geq 2$, is much simpler than their protocol, and removes some undesirable equilibria that arise in their protocol. Izmalkov, et al. [57] consider computation in which the players are colocated. This allows them to use certain physical assumptions, such as secure envelopes and ballot boxes, in addition to standard communication channels. In this environment, they demonstrate a protocol Π for securely implementing any mediated game Γ such that (informally) any equilibrium in Γ corresponds to an equilibrium in Π, and vice versa. Since rational secret sharing can be implemented as a mediated game, the work of [57] gives a solution to the problem we are considering here. Their work is in fact much more general, implying a protocol for any functionality, for arbitrary player utilities and even in the presence of coalitions. Concurrently with the work presented here, Abraham, et al. [1] defined a notion of resistance to *coalitions* of rational players and show a coalition-resistant protocol. Also concurrently, Lysyanskaya and Triandopoulos [70] examine the case of "mixed" security when *both* arbitrarily malicious and rational players might be present. Subsequent to our work, Kol and Naor [64] suggest using an alternative, stronger solution concept than the one we present below, called *strict Nash equilibrium*. They show how to achieve this using secret shares that are unbounded in size (though constant size in expectation), and demonstrate that this restriction is necessary.

This work and a second result by the same authors [63] were the first two works to remove the need for simultaneous broadcast channels (also discussed in Section 6.3). Ong et al. [75] consider the case where some players are guaranteed to be honest, while the rest are rational. They demonstrate a fair protocol for secret sharing in this setting, removing the simultaneous broadcast channel, and achieving a stronger solution concept than in prior work. Asharov and Lindell [2] consider the question of whether protocols for rational secret sharing can be independent of the players' utility functions, and demonstrate a negative result. They also consider whether it is possible to construct a protocol that does not require simultaneous broadcast, under the assumption that players might gain utility by forcing others to output incorrect values. They again demonstrate a negative result, though they do give the first protocol that achieves security when the utility for forcing bad output is known. Finally, the most recent work on the topic is by Fuchsbauer et al. [36] who use *verifiable random functions* to provide a much more efficient protocol than in prior work, by removing the need for general secure computation in the reconstruction phase. They also introduce two new solution concepts that address some of the issues we discuss in Section 6.3.

6.1 Definitions from Game Theory

A *game* consists of multiple players interacting according to chosen *strategies*. In our setting, strategies are simply probabilistic, polynomial-time interactive turing machines. We let σ_i denote the strategy employed by player P_i, and let $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ denote the vector of players' strategies. Following standard game-theoretic notation, we let $(\sigma_i', \vec{\sigma}_{-i}) \stackrel{\text{def}}{=} (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$; that is, $(\sigma_i', \vec{\sigma}_{-i})$ denotes the strategy vector $\vec{\sigma}$ with P_i's strategy changed to σ_i'.

Definition 6.1.1 *A vector of probabilistic, polynomial-time strategies $\vec{\sigma}$ is a* computational Nash equilibrium *if the following holds for all i: for any $\sigma_i' \neq \sigma_i$, we have $U_i(\sigma_i', \vec{\sigma}_{-i}) \leq U_i(\vec{\sigma}) + \mathsf{negl}(\kappa)$.*

That is, given that all other players are following $\vec{\sigma}_{-i}$, P_i cannot gain more than some negligible advantage by deviating and choosing some (efficient) strategy other than σ_i.

In general, multiple Nash equilibria may exist. An inherently "unstable" Nash equilibrium (i.e., one unlikely to be reached) is one in which any of the players' strategies are *weakly dominated* by other strategies. Informally, a strategy σ_i of player P_i is weakly dominated by another strategy σ_i' if (1) P_i is sometimes better off playing σ_i' than playing σ_i, and (2) P_i is never worse off playing σ_i' than playing σ_i. Recalling the example from the introduction, say a secret is shared using a t-out-of-n secret sharing (with $t < n$) and consider the strategy vector in which all n players reveal their shares. This is a Nash equilibrium: the secret is reconstructed even if any single player deviates. On the other hand, for each player P_i, revealing the share is weakly dominated by *not* revealing the share: if fewer than $t-1$ other players or more than $t-1$ other players reveal their shares, then nothing changes; if exactly $t-1$ other player reveal their shares then P_i learns the secret but no one else does. Formal definitions follow.

Definition 6.1.2 *Let S_i denote a set of strategies for P_i, and let $S_{-i} \stackrel{\text{def}}{=} S_1 \times \cdots \times S_{i-1} \times S_{i+1} \cdots \times S_n$. A strategy $\sigma_i \in S_i$ is* weakly dominated by a strategy $\sigma_i' \in S_i$ with respect to *S_{-i} if (1) there exists a $\vec{\sigma}_{-i} \in S_{-i}$ such that $U_i(\sigma_i, \vec{\sigma}_{-i}) < U_i(\sigma_i', \vec{\sigma}_{-i})$ and (2) for all $\vec{\sigma}_{-i} \in S_{-i}$, it holds that $U_i(\sigma_i, \vec{\sigma}_{-i}) \leq U_i(\sigma_i', \vec{\sigma}_{-i})$.*

Strategy σ_i is weakly dominated with respect to *S_{-i} if there exists a $\sigma_i' \in S_i$ such that σ_i is weakly dominated by σ_i' with respect to S_{-i}.*

Definition 6.1.3 *Let $\mathsf{DOM}_i(S_1 \times \cdots \times S_n)$ denote the set of strategies in S_i that are weakly dominated with respect to S_{-i}. Let S_i^0 denote the initial set of allowable strategies of P_i. For all $k \geq 1$, define S_i^k inductively as $S_i^k \stackrel{\text{def}}{=} S_i^{k-1} \setminus \mathsf{DOM}_i(S_1^{k-1} \times \cdots \times S_n^{k-1})$. Let $S_i^\infty \stackrel{\text{def}}{=} \cap_k S_i^k$.*

We say σ_i survives iterated deletion of weakly dominated strategies *if $\sigma_i \in S_i^\infty$.*

6.2 A Protocol for Rational Secret Sharing

We present our protocol under the assumption that the players have access to a simultaneous broadcast channel. In Section 6.3 we discuss subsequent work that removed this assumption [63, 64, 75, 2, 36]. We consider the case of arbitrary t, n, and we assume the dealer holds a secret s which lies in a *strict subset* S of a finite field $\mathcal{F}$ (if s lies in some field $\mathcal{F}'$, this is easy to achieve by taking a larger field $\mathcal{F}$ containing $\mathcal{F}'$ as a subfield). We assume players know S. Furthermore, we will assume that the shares of s are elements of $\mathcal{F}$, as in Shamir's (classical) secret sharing scheme.

The intuition behind the protocol is to use the threat that nobody will learn the secret in order to coerce cooperation and enable everybody to learn it. For the sake of presenting the intuition, assume the dealer is present during the reconstruction phase. (In our actual protocol, he is removed in the usual way through an execution of ShareGen.) At the beginning of each iteration, with probability α the dealer generates a random t-out-of-n Shamir sharing of s, and with probability $1-\alpha$ the dealer generates a random t-out-of-n Shamir sharing of an arbitrary element $\hat{s} \in \mathcal{F} \setminus S$; we describe how α is chosen below. These shares are distributed to the players. Note that no player can tell from their share whether the players were given a share of $\hat{s}$ or the true secret s. Then they all broadcast their shares simultaneously: if they recover some value $s \in S$, they know they have recovered the secret; they simply output s and terminate. If they recover some $\hat{s} \in \mathcal{F} \setminus S$, they return to the dealer to receive new shares and repeat the process. If a player ever fails to broadcast, all players abort the protocol. As described previously, if a player tries to hold back, hoping to reconstruct the secret alone, they run the risk of aborting in a round when nothing is learned, forcing the protocol to terminate before they can recover the secret. By choosing α appropriately, we can prove that they have higher expected utility when they cooperate.

Theorem 6.2.1 *Given an ideal execution of* ShareGen*, and for appropriate choice of α, the above protocol constitutes a Nash equilibrium for t-out-of-n secret sharing that survives iterated deletion of weakly dominated strategies.*

Proof: We first consider the case of $t = n = 2$, and then discuss how to generalize the proof for arbitrary t, n. It is not hard to see that the protocol is a Nash equilibrium for appropriate choice of α: Say P_2 acts according to the protocol and consider whether P_1 has any incentive to deviate. Without loss of generality, consider a deviation in the first iteration. The only possible deviation is for P_1 to refuse to broadcast his share. In this case, he learns the secret (while P_2 does not) with probability α, but with probability $1-\alpha$ he will never learn the secret.

Say P_1's utility is U^+ if he learns the secret but P_2 does not; U if both players learn the secret; and U^- if neither player learns the secret, where $U^+ > U > U^-$. If P_1 follows the

ShareGen

Inputs: Let the inputs to ShareGen be $((pk_1, s_1, \sigma_1), \ldots, (pk_n, s_n, \sigma_n))$. The security parameter is κ. If any of the following statements are true, output $\perp$ and terminate:

- If for some j it holds that $s_j \notin \mathcal{F}$.
- If for some i, j it holds that $pk_i \neq pk_j$.
- If for some j it holds that $\mathsf{Vrfy}_{pk_j}(s_j, \sigma_j) = 0$.

Computation:

For $1 \leq i \leq m$, define values $s_1^{(i)}, \ldots, s_n^{(i)}$ in the following way:

- Choose i^* according to a geometric distribution with parameter α (see text).
- For $i = 1$ to $i^* - 1$ do:
  - Choose $\hat{s} \in \mathcal{F} \setminus S$ and set $(s_1^{(i)}, \ldots, s_n^{(i)}) = \mathsf{Share}(t, n, \hat{s})$.
- For $i = i^*$ to m, set $(s_1^{(i)}, \ldots, s_n^{(i)}) = \mathsf{Share}(t, n, s)$.

Output:

1. Send to P_j the values $(s_j^{(1)}, \ldots, s_j^{(m)})$

Figure 6.1: Functionality ShareGen for rational secret sharing.

protocol, his expected utility is U. If P_1 deviates, his expected utility is $\alpha \cdot U^+ + (1 - \alpha) \cdot U^-$. So as long as

$$U > \alpha \cdot U^+ + (1 - \alpha) \cdot U^- ,$$

it is in P_1's best interest to follow the protocol. For appropriate $\alpha \in (0, 1)$, then, the strategy profile in which both parties follow the protocol is a Nash equilibrium.[1] It is immediate that the same analysis holds for general t, n, regardless of the number of participating players t^*.

We next prove that our protocol survives iterated deletion of weakly dominated strategies. It is easy to see that when a player learns the real secret, *not* aborting afterwards is weakly dominated by aborting. The first round of iterated deletion thus leaves only strategies in which players always abort after learning the secret.

We show that no other deterministic strategies are weakly dominated (and hence no randomized strategies are weakly dominated either). We again begin with the case $t = n = 2$. We show that for all deterministic strategies σ, σ' of P_1, there exist strategies τ, τ' of P_2 such that $U_1(\sigma, \tau) > U_1(\sigma', \tau)$ but $U_1(\sigma, \tau') < U_1(\sigma', \tau')$. This proves that all deterministic strategies of P_1 are incomparable, and so none are ever deleted.

Let $h_i(\sigma, \tau; r)$ denote the history of actions (by both players) through iteration i given the indicated strategies σ and τ and assuming the dealer uses coins r (we view r as an infinite string encoding the dealer's random choices in all iterations). $h_0(\sigma, \tau; r)$ denotes the empty (starting) history. Let $A_i(\sigma, \tau; r)$ denote the action taken by P_1 in iteration i, again for the indicated strategies and random coins. We say a player *cooperates* in some iteration if they reveal their share, and *defects* if they do not.

Now take arbitrary deterministic strategies $\sigma \neq \sigma'$ for P_1. Let τ^0 be a strategy of P_2, $i \geq 1$

[1] We intentionally designed the protocol such that it can run forever, though with overwhelming probability in κ it will end in the first κ rounds. Without this specification, the protocol would not technically survive iterated deletion of weakly dominated strategies. Even though it occurs negligibly often, the players will have no incentive to broadcast when they reach the last round. Then, through inductive reasoning, it follows that they will choose not to broadcast even in the first round. See further discussion in Section 6.3.

Protocol for Rational Secret Sharing

Sharing Protocol:

Inputs: The dealer has input $s \in S$. The security parameter is κ.

Let $(\mathsf{Share}, \mathsf{Rec})$ be some classical secret sharing scheme, and $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a digital signature scheme.

1. The dealer computes $(s_1, \ldots, s_n) = \mathsf{Share}(t, n, s)$, and $(sk, pk) = \mathsf{Gen}(1^\kappa)$.
2. The dealer gives $(pk, \mathsf{Sign}_{sk}(s_i))$ to P_i.

Reconstruction Protocol:

Inputs: Party P_j has input (pk, s_j, σ_j).

The protocol:

1. **Preliminary phase:**
   (a) The parties run a protocol π for computing $\mathsf{ShareGen}$. Each player P_j uses their respective input, s_j and security parameter κ.
   (b) Any player that receives $\perp$ from this execution aborts the protocol. Otherwise, denote P_j's output by $(s_j^{(1)}, \ldots, s_j^{(\kappa)})$ and continue to the next stage.
2. **For $i = 1, \ldots, \kappa$ do:**
   Broadcast shares:
   (a) Each P_j simultaneously broadcasts $s_j^{(i)}$.
   (b) If anyone fails to broadcast, or broadcasts an invalid share, all players abort.
   (c) Otherwise, for $j \in \{1, \ldots, n\}$, let $(s_1^{(i)}, \ldots, s_{j-1}^{(i)}, s_{j+1}^{(i)}, \ldots, s_n^{(i)})$ denote the values received by P_j.
      i. If $\mathsf{Rec}(s_1^{(i)}, \ldots, s_{j-1}^{(i)}, s_j, s_{j+1}^{(i)}, \ldots, s_n^{(i)}) \in S$, P_j outputs s and terminates the protocol.
      ii. If $\mathsf{Rec}(s_1^{(i)}, \ldots, s_{j-1}^{(i)}, s_j, s_{j+1}^{(i)}, \ldots, s_n^{(i)}) \in \mathcal{F} \setminus S$, P_j proceeds to the next round.
3. **Repeat the reconstruction protocol:**
   With all but negligible probability, the protocol will have ended before now. If it has not, the players restart the reconstruction using the same inputs.

Figure 6.2: A protocol for rational secret sharing.

be an integer, and r be a set of coins such that

$$h_{i-1}(\sigma, \tau^0; r) = h_{i-1}(\sigma', \tau^0; r) \tag{6.1}$$

but

$$A_i(\sigma, \tau^0; r) \neq A_i(\sigma', \tau^0; r); \tag{6.2}$$

i.e., iteration i is the first iteration in which the actions of P_1 differ when we compare strategies σ and σ' as played against τ^0. (Note that some such τ^0, i, r must exist or else $\sigma = \sigma'$. Also, it is implicit that neither player has reconstructed the real secret in any prior iteration, since otherwise the protocol would never reach iteration i.) Without loss of generality, assume $A_i(\sigma, \tau^0; r)$ is to defect and $A_i(\sigma', \tau^0; r)$ is to cooperate. Note that the actions in iteration i cannot depend on whether or not the dealer shared the real secret or the fake secret in that iteration.

Consider the following strategy τ of P_2: (1) act identically to τ^0 through iteration $i - 1$;

(2) in iteration i, defect; (3) in all subsequent iterations: if P_1 defected in iteration i, then cooperate; if P_1 cooperated in iteration i, defect. For any r satisfying Equations (1) and (2), P_1 is clearly better off playing σ than σ' against strategy τ.

Next consider the following strategy τ' of P_2: (1) act identically to τ^0 through iteration $i-1$; (2) in iteration i, cooperate; (3) in all subsequent iterations: if P_1 defected in iteration i, then defect; if P_1 cooperated in iteration i, cooperate. Exactly as when we argued earlier that our protocol was a Nash equilibrium, we have $U_1(\sigma, \tau') < U_1(\sigma', \tau')$. (Here, P_1 is not better off playing σ' than σ for *all* r satisfying Equations (1) and (2); however, P_1 *is* better off playing σ' in expectation.)

The same argument extends to the case of general t, n, regardless of the number of participating players t^*. We simply replace τ^0 with a strategy profile of $n-1$ strategies such that Equations (6.1) and (6.2) above are still valid, and then define τ and τ' as above, but modifying the strategies of all other players. ∎

We remark that when $t^* = t$ our protocol has no additional Nash equilibrium which is preferred, by any player, to the prescribed equilibrium.

6.3 Discussion

Solution concepts: The work of Kol and Naor [63] later demonstrated a further subtlety that arises when using the notion of iterated deletion as a solution concept. Recall that in Theorem 6.2.1, we assumed an ideal execution of ShareGen. If we replace this with a secure computation, where the execution makes use of cryptographic keys of size κ (for oblivious transfer), then if the protocol ever reaches the 2^κth iteration, a player that has been trying to break the security of these schemes is guaranteed to succeed by that time, and might fully recover their opponent's shares. Even though this happens with negligible probability, logically, if it does happen, there is no reason for anyone to broadcast at that point, since their opponents may have already recovered the secret and are not going to broadcast their own shares. Now, the protocol is susceptible to backwards induction: if one player knows that the others will not send their share in round 2^κ, then logically, they should refuse to broadcast in round $2^\kappa - 1$, and so forth. Following the logic to its conclusion, nobody will choose to broadcast even in the first round! The point is, while finding the secret keys used in the secure computation may be computationally infeasible, the players are not bounded in their inductive reasoning.

Kol and Naor offer solutions to the problem by introducing techniques that are secure in an information theoretic sense up until the round in which the secret is revealed [63, 64]. However, perhaps the more important point to take from their work is that the standard game theoretic definitions do not necessarily translate nicely to the setting where players are computationally bounded. In this particular case, it seems that the problem lies more in the solution concept rather than in the protocol. One possible solution might be to introduce a *discount factor* in the utility function, decreasing the payoff for any benefit that occurs far in the future, and ensuring that the advantage to aborting in round 2^κ is negligible anyway.

There is a separate problem with the solution concept used in Section 6.2. We argued there that because the prescribed strategies are not weakly dominated, the preferred equilibrium is more likely to occur. However, we proved that (essentially) *no* strategies are weakly dominated, which certainly waters down the argument that our preferred strategy is the most likely outcome. Even worse is that we used an *empty threat* to prove the theorem, by considering a strategy that cooperates forever if the other player defects! As opposed to the concern that was addressed by Kol and Noar, the problem here is not unique to the computational set-

ting; the problem lies in the very definition of the solution concept, which predates the study of game theory in cryptography. However, if we try adopting some of the stronger solution concepts from the literature instead, we run into difficulties in the computational setting. For example, we might prefer to require that our prescribed strategy is *sub-game perfect*, which states that the prescribed strategy must be a Nash equilibrium at every possible state in the game tree, even those that are never reached when the strategy is correctly followed. Indeed, this definition is designed to remove empty threats such as the one described above. Unfortunately, in the computational setting, this requires us to consider even the histories in which players happen to guess a cryptographic key. As a result, natural protocol descriptions are unlikely to meet the definition as is.

A final point on our solution concept was mentioned in a personal conversation with Amos Beimel. Technically, our proof in Section 6.2 only considers strategies that take the action-histories of the other players as input. But a strategy could also consider the particular message strings sent by the other players (including the signatures on their secret shares). For example, although it is intuitively irrational, a strategy could dictate that a player aborts if the third bit of the signature received from player j in the prior round is 0. When we consider such strategies, it is no longer clear that the prescribed strategy survives iterated deletion of weakly dominated strategies. However, Beimel demonstrated that it is not very difficult to modify our theorem to prove that all *envelope* strategies survive iterated deletion, where these are the strategies that we considered, but never defined, in which the only input to the strategy is the action-histories.

Simultaneous broadcast: We assumed here that the players have access to a simultaneous broadcast channel. This is not always a realistic assumption, especially if protocols are executed over the Internet, and it is best if it can be avoided. The works of Kol and Naor were the first to remove this assumption [63, 64], followed by Ong et al. [75], Asharov and Lindell [2], and finally Fuchsbauer et al. [36]. With the exception of Ong et al. (who rely on the assumption that some of the players are fully honest, rather than rational), all of these works use the same fundamental idea (though most of them implicitly). If the parties do not know enough about the secret that they can recognize it when they see it, then we can safely instruct them take turns revealing their shares. One player will learn the secret first, but they will not know that they have learned it until *after* they have responded with their own share, enabling the other party to learn the secret as well. Then, only in the following round are the players informed that the value they previously learned is the correct secret.

General secure computation: All of the above work deals exclusively with secret sharing. A natural next question is whether we can extend this work to enable fairness in general secure computation among rational parties. The naive approach would be to have the players compute an unfair secure computation that results in a secret sharing of the outputs, and then to apply the fair reconstruction protocol from any of the above works. Unfortunately, without simultaneous broadcast, some complications arise. First is the issue described above regarding prior knowledge of the output. For certain applications of secret sharing, it might be reasonable to assume that the secret is chosen uniformly from the domain of all secrets. However, for general secure computation, we do not want to be constrained to assuming that the inputs are uniformly chosen. Furthermore, even if we do make this assumption, for more general computations, the input itself may give too much information for us to assume that he output will be unrecognized. For example, consider the case of signature exchange where the player is given his opponents verification key as part of his input. As soon as one player reconstructs a secret that verifies, he can abort the protocol and ruin fairness.

We might hope that the naive solution above will work if we are given a simultaneous broadcast channel, but another subtlety still needs to be addressed. Suppose two players are computing the XOR function on inputs b_1 and b_2. If player one is rational (with the utility functions described earlier), he will switch his input value to $\bar{b}_1$, causing player two to output the wrong value; he can then still output the correct value by flipping his own output bit from the computation! This is considered a legal action in the standard definition of security (among malicious players): we always allow the adversary to change inputs before submitting values the the trusted party. But in the rational setting we still have no way to model this.

In summary, while our work helped to introduce the question of rational computation (along with [53, 70, 1]), it brings with it some key definitional questions. Many of these questions are still open today. We refer the reader to a more detailed discussion in a survey by Katz [59] for futher information.

Chapter 7

Fair Primitives for Secure Computation

In this paper we address a very natural question. What is the *minimum* amount of help required to be able to compute all functions fairly? We think of this helper as a naive black box, or a *primitive*, with no knowledge of the function being computed. It is charged with a fixed task: it takes inputs from each player, and then simultaneously outputs some fixed function of the inputs to all players. Clearly we can compute any function fairly if this primitive is sufficiently complex: we can simply define its input to be a description of the function being computed, along with the inputs to that function. (Indeed, this was demonstrated by Fitzi et al. [35], as discussed below.) However, our interest is in reducing the complexity of the primitives. In particular, we study the minimum input size to such primitives that will enable the fair computation of any function.

Interestingly, there has been extensive research on very similar questions in the context of *unfair* secure computation. When there is no honest majority among the players, it is known that oblivious transfer is both necessary and sufficient for computational security (without fairness) [89, 46, 62, 56]. There is a long line of research identifying the minimum primitives that enable information theoretic security in this setting [24, **?**]. Surprisingly, very little work has addressed the parallel questions with respect to fairness. One exception is the work of Lepinski, Micali, Peikert, and shelat [67], where the authors devised a protocol for completely fair multi-party computation with any number of malicious parties by relying on "envelope" primitives: communication primitives with special physical properties. The goal of their work was to design a fair protocol that makes use of easily realizable primitives, rather than to explore the bounds on the complexity of these primitives or the number of required interactions with the primitive, as we do below.

7.1 A Complete Primitive for Fair Two-Party Computation

In this section we demonstrate a primitive that is complete for two-party fairness. (In the original paper [50] we also provide a similar primitive for the multi-party setting, but we do not present it here.) In order to compute some function $\mathcal{F}(x, y) = \{f_\kappa^1(x, y), f_\kappa^2(x, y)\}$ fairly, the parties will first unfairly compute a related function $\mathcal{F}'(x, y)$ that provides player i with an *encryption* of $f_\kappa^i(x, y)$, along with a secret share of the corresponding decryption key (generated using a 2-out-of-2 non-malleable secret-sharing scheme, defined below). This reduces the problem of fairness to a simple exchange of the secret shares. Of course, if the players exchanged these on their own, one player might abort just at the point of exchange, recovering the decryption key (and thus his output) all alone. Instead, the ideal functionality

FairRec (described next) takes the shares from each player and performs the reconstruction fairly; the non-malleability property of the secret-sharing scheme enables the functionality to verify thzat both players have provided correct shares. The details follow. To simplify the notation, we drop the security parameter when it is not important to the discussion.

Definition 7.1.1 (Fair Reconstruction) $\mathsf{FairRec}_\ell(x, y)$ *takes inputs of size* $|x| = |y| = \ell$ *and outputs:*

$$\mathsf{FairRec}_\ell(x, y) = \begin{cases} (s, s) & \text{if } \mathsf{Rec}(x, y) = (s, 0) \\ (\perp, \perp) & \text{otherwise} \end{cases}$$

where Rec *is the reconstruction protocol from a non-malleable secret sharing scheme.*

Intuitively, the FairRec functionality is just a fair implementation of Rec: it takes a non-malleable secret share from each player, and outputs the result of Rec to both players if and only if the secret was successfully reconstructed. We will prove that it is complete for fairness in Section 7.1. Interestingly, it will also play a key role in our proofs of impossibility in Section 7.2.

Theorem 7.1.2 *Assuming one-way functions exist, any two-party functionality* $\mathcal{F}$ *can be fairly computed in the OT-hybrid model by using a single call to* $\mathsf{FairRec}_{O(\kappa)}$.

Proof: We begin by defining a function $\mathcal{F}'$ related to $\mathcal{F}$ in the way described above. Specifically, let $(\mathsf{Enc}, \mathsf{Dec})$ be the encryption and decryption functions for a semantically secure encryption scheme. Then we define:

$$\mathcal{F}'(x, y) = \left\{ f_\kappa'^1(x, y), f_\kappa'^2(x, y) \right\} = \left\{ \left(\mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^1(x, y)), s_1 \right), \left(\mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^2(x, y)), s_2 \right) \right\}$$

where $(s_1, s_2) = \mathsf{Share}(k_{\mathsf{Enc}}; r)$, r is chosen uniformly at random, and $|k_{\mathsf{Enc}}| = \kappa$.

From standard results in cryptography, we can build a $(2, 2)$-NMSS scheme unconditionally. Furthermore, by the results of [61], any two-party functionality can be computed securely *with abort* given black box access to OT. In particular, the following lemma follows from previous work:

Lemma 7.1.3 *Assuming the existence of oblivious transfer, for any function* $\mathcal{F}(x, y)$*, there exists a two-party protocol* $\Pi'(x, y)$ *that securely computes* $\mathcal{F}'(x, y)$ *with abort.*

The size of the input to FairRec is the size of the share of one decryption key. Fair computation of $\mathcal{F}(x, y)$ follows easily:

1. Execute a secure-with-abort protocol to compute $\mathcal{F}'(x, y)$. The existence of such a protocol follows from Lemma 7.1.3.

2. Player $i \in \{1, 2\}$ parses the output $f_\kappa'^i(x, y)$ as

   $$z_i = (z_{\mathsf{Enc}}, z_{\mathsf{FairRec}}) = \left(\mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^i(x, y)), \mathsf{Share}(k_{\mathsf{Enc}}, r)_i \right)$$

   and submits z_{FairRec} to the ideal function FairRec.

3. Let k_i denote the output that player i receives from FairRec. If $k_i = \perp$, output $\perp$. Otherwise, output $\mathsf{Dec}_{k_i}(z_{\mathsf{Enc}})$.

To demonstrate that the resulting computation is secure, we prove security in the hybrid world where we are given a semantically-secure symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$, an ideal functionality for computing FairRec with perfect fairness, and an ideal functionality for computing $\mathcal{F}'(x, y)$ with abort. Let κ also denote the security parameter for the NMSS and encryption schemes (i.e., no polynomial-time adversary has advantage more than $\epsilon = \mathsf{negl}(\kappa)$ in the semantic-security game).

Note that if we allow the input size to FairRec to depend on the output size of $\mathcal{F}$, we do not need to use encryption at all: $\mathcal{F}'$ can directly output shares of $\mathcal{F}$ and FairRec can be used to exchange these shares. In this case, the proof of security is also much simpler and uses a *straight-line* simulator. The ideal-world simulator in this proof is a little more complex and uses its ability to rewind the adversary:

The simulator in the ideal world has access to an ideal functionality for computing $\mathcal{F}(x, y)$ with perfect fairness, and simulates FairRec and $\mathcal{F}'$ for the real-world adversary. The simulator works as follows (we assume without loss of generality that the adversary has corrupted player 1):

1. $\mathcal{S}$ receives input x from player 1 intended for $\mathcal{F}'$. He simulates $\mathcal{F}'$ as follows:
   (a) $\mathcal{S}$ generates a random encryption key k_{Enc}.
   (b) $\mathcal{S}$ computes a secret sharing of k_{Enc} by randomly choosing r and running $(s_1, s_2) \leftarrow \mathsf{Share}(k_{\mathsf{Enc}}, r)$.
   (c) $\mathcal{S}$ computes $\xi = \mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^1(x, 0))$.
   (d) $\mathcal{S}$ sends (ξ, s_1) to player 1.
2. $\mathcal{S}$ receives input s' from player 1 intended for FairRec. If player 1 sends any value $s' \neq s_1$ (including $s' = \perp$), we refer to this as an abort.
3. If player 1 aborted in the previous step, $\mathcal{S}$ returns $\perp$ on behalf of FairRec, submits $\perp$ to $\mathcal{F}$, outputs the view of player 1, and terminates the simulation.
4. Otherwise, $\mathcal{S}$ will try to estimate the probability that player 1 does not abort in step 2. (This is done exactly as in Goldreich et al. [43].) We denote the true probability of this event by q, where the probability is taken over the randomness used in generating k_{Enc} (Step 1a), and in generating the encryption (Step 1c). We denote S's estimate of q by $\tilde{q}$.
   (a) $\mathcal{S}$ fixes some number $t = \mathrm{poly}(\kappa) \geq \kappa$. (Its value is described more precisely below.)
   (b) $\mathcal{S}$ rewinds player 1 and executes steps 1 and 2 repeatedly, using fresh randomness each time, until player 1 has not aborted in t of the repetitions.
   (c) $\mathcal{S}$ estimates q as $\tilde{q} = t/(\#\text{ of repetitions})$. The polynomial defining t is chosen to be large enough that $\Pr[\frac{1}{2} \leq \frac{q}{\tilde{q}} \leq 2] > 1 - 2^{-\kappa}$.
5. The simulator sends x to the ideal functionality for $\mathcal{F}$ and receives $f_\kappa^1(x, y)$. $\mathcal{S}$ then repeats the following procedure at most $\min(\frac{t}{\tilde{q}}, 2^\kappa)$ times.
   (a) Using fresh randomness, he rewinds player 1 and repeats Steps 1 and 2, using $\xi = \mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^1(x, y))$, in place of $\xi = \mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^1(x, 0))$.
   (b) If player 1 does not abort, $\mathcal{S}$ simulates the output of FairRec by sending k_{Enc} to player 1. He outputs the transcript that successfully completed, and terminates the simulation.

6. If $\mathcal{S}$ has not yet terminated, it outputs fail and aborts the simulation.

It is easy to observe that when the simulator does not output fail, the hybrid world and the ideal world are identical. We next bound the probability that the simulator outputs fail, and afterward we will prove that the simulator runs in expected polynomial time.

Claim 7.1.4 *The simulator outputs* fail *with probability that is negligible in* κ.

Proof: Note that $\mathcal{S}$ outputs fail only if a) the adversary did not abort at step 2, and b) the simulator made $t/\tilde{q}$ attempts in step 5 and did not succeed in producing a transcript. Let p be the probability that player 1 does not abort in step 2 given $\xi = \mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^1(x,y))$ (as opposed to q which is the probability that he does not abort given $\xi = \mathsf{Enc}_{k_{\mathsf{Enc}}}(f_\kappa^1(x,0))$). The following analysis is taken from [43]:

$$\begin{aligned}
\Pr[\mathcal{S} \text{ outputs fail}] &= q\sum_i \left(\Pr\left[\frac{1}{\tilde{q}} = i\right]\right)(1-p)^{t\cdot i} \\
&\leq q\Pr\left[\frac{q}{\tilde{q}} \geq \frac{1}{2}\right](1-p)^{t/\tilde{q}} + q\Pr\left[\frac{q}{\tilde{q}} < \frac{1}{2}\right] \\
&\leq q(1-p)^{t/2q} + \mathsf{negl}(\kappa) \qquad (7.1)
\end{aligned}$$

We wish to prove that the last equation is negligible in κ. Let's suppose first that $p \geq q/2$. It follows then that

$$(1-p)^{t/2q} \leq (1-\frac{q}{2})^{t/2q} < e^{-t/4}$$

On the other hand, if $p < q/2$ and Equation (7.1) is non-negligible, i.e, for some polynomial g and infinitely many values κ:

$$q(1-p)^{t/2q} + \mathsf{negl}(\kappa) > 1/g(\kappa),$$

then it follows that $q > 1/g'(\kappa)$ for some polynomial g'. Finally, we have $|q-p| > q/2 > 1/(2g'(\kappa))$, violating the semantic security of the encryption scheme. ∎

Claim 7.1.5 *The simulator's expected running time is polynomial in* κ.

Proof: First, note that if the adversary fails at step 2, the simulation ends immediately; this occurs with probability $1-q$ (the probability is over the random coins used by the simulator).

With probability q, the simulator runs the estimation process of Step 4, followed by the iterations of Step 5. The iterations in either step are polynomial time, so we will denote an upper-bound on their runtime by $g(\kappa)$. In the estimation process, the probability that it succeeds in any one iteration is q, and it runs this process until it has succeeded $t = \mathrm{poly}(\kappa)$ times. The expected number of iterations is therefore t/q. In the iteration process, the number of iterations is bounded by $\min(\frac{t}{\tilde{q}}, 2^\kappa)$. With probability greater than $1-2^{-\kappa}$, we have $t/\tilde{q} < 2t/q$, and with the remaining probability, the runtime is bounded at 2^κ. Hence the total expected run time is bounded by

$$g(\kappa)\cdot q(\frac{t}{q} + \frac{2t}{q}) = 3tg(\kappa) = \mathrm{poly}(\kappa)$$

∎
∎

7.2 A Lower Bound on the Size of Complete Primitives

In this section we show that there does not exist a finite (i.e. "short") primitive that is complete for fairness. More specifically, we prove that the $\mathsf{FairRec}_\kappa$ function cannot be fairly computed even if the players are given parallel access to a primitive of size $O(\log \kappa)$. There are two main ideas behind the proof. For simplicity, imagine for now that the entire protocol consisted of a single call to this short primitive. Our first observation is that because the primitive is short, the adversary can locally simulate it, computing its output for each possible input of the other party. This will play a crucial role in our proof, but it does not itself suffice: so far the adversary has no way of knowing *which* of these outputs are correct. However, because the primitive is supposed to be complete for fairness, it allows us to compute the FairRec functionality, which has a very useful property: its output is *verifiable*. That is, when two parties are given inputs generated by Share, then the correct output of FairRec is $(s, 0)$, where the flag 0 indicates that s is the correct output. Furthermore, for incorrect inputs, with overwhelming probability the output of FairRec is $(\perp, \perp)$. The adversary simply computes the primitive for every possible input of the other player, and outputs s when he recovers it.

When we consider a protocol with many calls to the primitive (including parallel calls), we combine the above ideas using a standard hybrid argument. If the adversary aborts before any invocations of the primitive, he cannot learn anything about the output s. On the other hand, if he behaves honestly in all invocations, he should always recover s. We prove below that there is some specific invocation for which the adversary can gain a non-negligible advantage over the honest party by aborting and simulating the input to that invocation as described above. Finally, he can guess which invocation will allow this advantage with significant probability. Formally, we define:

Definition 7.2.1 (Parallel Primitives) *For a primitive g, we denote $\mathsf{par}_k(g)$ the primitive that consists of k independent copies of g with* enforced parallelism. *The parallelism is enforced in that none of the copies of g in $\mathsf{par}_k(g)$ send output to any party until all k copies have received input from all parties. We use Π_p^g to denote that protocol Π has access to $\mathsf{par}_k(g)$.*

Note that for any $k \geq 1$, $\mathsf{par}_k(g)$ is a more powerful primitive than g (i.e., if the fairness of $\mathcal{F}$ reduces to g then the fairness of $\mathcal{F}$ also reduces to $\mathsf{par}_k(g)$). We are proving an impossibility, so starting with a more powerful primitive strengthens our results. Our proof will hold even if we restrict the adversarial behavior to aborting early.

Theorem 7.2.2 *Let g be an $O(\log \kappa)$-bit primitive. Then for any polynomial p, $\mathsf{par}_{p(\kappa)}(g)$ is not complete for fairness.*

Proof: Suppose there exists such a primitive g and polynomial p. Consider the $r = r(\kappa)$ round protocol $\Pi_p^{\vec{g}}$ that fairly computes $\mathsf{FairRec}_\kappa(x, y)$ while making a call to $\mathsf{par}_{p(\kappa)}(g)$ in each round. We can think of this call as $p(\kappa)$ parallel calls to g. Without loss of generality, we assume that these calls to g constitute the only communication between the players[1]. Let $q = p \cdot r$ be the total number of calls to g. For each round $i \in r$, we define some arbitrary ordering σ_i on the parallel calls to g that occur in that round. This induces a natural ordering over all q calls to g, where for $i < j$, calls in round i are ordered before calls in round j. We let g_k denote the k^{th} call to g according to this ordering.

[1]This is without loss of generality because we can always modify g to do message transmission, in addition to its original functionality. Note also that if less than $p(\kappa)$ calls are needed in a particular round, the players can make extra calls with random inputs, ignoring the outputs, to make the total number of calls $p(\kappa)$.

Consider an execution of $\Pi_p^{\vec{g}}$ in which the players' inputs are generated by running $\mathsf{Share}(s,r) = (s_0, s_1)$ for random s and r, and player j gets the share s_j. We let the value a_i denote the output of player 0 when player 1 acts honestly for the first i calls to g (according to the ordering previously described) and then aborts. We define b_i in the symmetric way. Note that by correctness of $\Pi_p^{\vec{g}}$, and the definition of $\mathsf{FairRec}_\kappa$, for all i

$$\Pr[a_i \neq s \wedge a_i \neq \perp] = \mathsf{negl}(\kappa) = \Pr[b_i \neq s \wedge b_i \neq \perp]$$

and

$$\Pr[a_q = s] = \Pr[b_q = s] = 1 - \mathsf{negl}(\kappa).$$

where the probability is over the random tapes of the players. Furthermore, by the definition of FairRec and the properties of a NMSS scheme,

$$\Pr[a_0 \neq \perp] = \mathsf{negl}(\kappa) = \Pr[b_0 \neq \perp].$$

It follows that for every large enough κ, there exists a polynomial $p'(\kappa)$ and a round i such that either

$$\Pr[a_i = s] - \Pr[b_{i-1} = s] \geq \frac{1}{p'(\kappa)}.$$

or

$$\Pr[b_i = s] - \Pr[a_{i-1} = s] \geq \frac{1}{p'(\kappa)}.$$

Without loss of generality, we will assume the former, and we demonstrate an adversary $\mathcal{A}$ that breaks the security of $\Pi_p^{\vec{g}}$ with probability at least $1/(q \cdot p'(\kappa))$.

$\mathcal{A}$ begins by choosing a random value $i^* \in [1, \ldots, q]$, and plays honestly for the first $i^* - 1$ calls to g (i.e., submits correct values to g) and then aborts. Note that the resulting output of player 1 is b_{i^*-1}. The adversary now attempts to compute the value of a_{i^*} by simulating the side of player 1. Note, however, that by definition, the value of a_{i^*} depends on honest input to g_{i^*} from both players, and $\mathcal{A}$ may not know (anything) about player 1's input to g_{i^*}. Here we use the fact that g has short inputs, and that FairRec is verifiable. $\mathcal{A}$ goes through all possible inputs $\beta \in \{0,1\}^{O(\log \kappa)}$ that player 1 might have sent to g_{i^*}, and for each such value he simulates g internally, using as input his own (honest) value that he *would* have sent if he had not aborted, and β. He computes a_{i^*} from his view in the (real) interaction with player 1, and the simulated output of g_{i^*}. Since one of these values of β is the value used by player 1 in the actual execution, it follows that the correct value of a_{i^*} is among this set of outputs. Furthermore, if some simulated $a_{i^*} = s' \neq \perp$ then $s' = s$ with overwhelming probability. $\mathcal{A}$ outputs $s' \neq \perp$ if this occurs, and $\perp$ otherwise. By our assumption, there exists an i such that $\Pr[a_i = s \wedge b_{i-1} = \perp] \geq \frac{1}{p'(\kappa)}$. Hence, $\mathcal{A}$ recovers s without the honest party receiving output with probability $1/(q \cdot p'(\kappa))$, contradicting the fairness of protocol. ■

Corollary 7.2.3 *Simultaneous broadcast is not complete for fairness.*

Proof: The fact that short simultaneous broadcast is not complete for fairness follows from Theorem 7.2.2. We prove now that long simultaneous broadcast can be simulated given parallel access to a short simultaneous broadcast protocol; it follows that if long-SB is complete, short-SB must also complete, contradicting Theorem 7.2.2.

Let g denote the k-bit SB primitive. Then for any $p \in \mathbb{N}$, there exists a protocol $\Pi_p^{\vec{g}}$ that implements kp-bit SB with perfect security, given p parallel copies to g. The protocol is the

(trivial) one round protocol in which both parties split their inputs into p blocks of size k, submit block i to instance g_i, and output the concatenation of the p outputs (maintaining the order). The proof of the lemma is straightforward, so we omit a formal exposition. We simply note that the decision of an adversary to change its input to any instance(s) of g (including the decision to abort in some instance(s)) is entirely independent of the actions or input of the honest party. The simulator simply recovers the p values that the adversary intended for g (recall that an abort is treated as input 0^k), concatenates them, and forwards them to the trusted party. After receiving output, the simulator rewinds the adversary, parses the output into blocks, and sends block i to the adversary as though it were the honest player's input to g_i. ■

Bibliography

[1] Ittai Abraham, Danny Dolev, Rica Gonen, and Joseph Y. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *PODC*, pages 53–62, 2006.

[2] Gilad Asharov and Yehuda Lindell. Utility dependence in correct and fair rational secret sharing. In *CRYPTO*, pages 559–576, 2009.

[3] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *ACM Conference on Computer and Communications Security*, pages 7–17, 1997.

[4] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *EUROCRYPT*, pages 591–606, 1998.

[5] Giuseppe Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *ACM Conference on Computer and Communications Security*, pages 138–146, 1999.

[6] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*, pages 137–156, 2007.

[7] Baruch Awerbuch, Manuel Blum, Benny Chor, Shafi Goldwasser, and Silvio Micali. How to implement bracha's O(log n) Byzantine agreement algorithm. Unpublished, 1985.

[8] Feng Bao, Robert H. Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[9] Donald Beaver. Multiparty protocols tolerating half faulty processors. In *CRYPTO*, pages 560–572, 1989.

[10] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

[11] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *CRYPTO*, pages 589–590, 1989.

[12] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts (extended abstract). In Wilfried Brauer, editor, *ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 43–52. Springer, 1985.

[13] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[14] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press.

[15] Manuel Blum. How to exchange (secret) keys. *ACM Trans. Comput. Syst.*, 1(2):175–193, 1983.

[16] Dan Boneh and Moni Naor. Timed commitments. In *CRYPTO*, pages 236–254, 2000.

[17] Ernest F. Brickell, David Chaum, Ivan Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *CRYPTO*, pages 156–166, 1987.

[18] Andrei Z. Broder and Danny Dolev. Flipping coins in many pockets (Byzantine agreement on uniformly random values). In *FOCS*, pages 157–170, 1984.

[19] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *CRYPTO*, pages 93–111, 2000.

[20] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

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

[22] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[23] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395. IEEE, 1985.

[24] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.

[25] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[26] Richard Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *CRYPTO*, pages 573–588, 1989.

[27] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation, an introduction. http://www.cs.au.dk/~jbn/smc.pdf, 2009.

[28] Ivan Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. Cryptology*, 8(4):201–222, 1995.

[29] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[30] Yevgeniy Dodis, Pil Joong Lee, and Dae Hyun Yum. Optimistic fair exchange in a multi-user setting. In *Public Key Cryptography*, pages 118–133, 2007.

[31] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[32] Shimon Even. Protocol for signing contracts. In *CRYPTO*, pages 148–153, 1981.

[33] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *CRYPTO*, pages 205–210, 1982.

[34] Shimon Even and Yacov Yacobi. Relations among public key signature schemes. Technical Report #175, Technion Israel Institute of Technology, Computer Science Department, 1980. http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/1980/CS/CS0175.

[35] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.

[36] Georg Fuchsbauer, Jonathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In *TCC*, pages 419–436, 2010.

[37] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *CRYPTO*, pages 135–155, 1987.

[38] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography*, pages 168–182, 2002.

[39] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO*, pages 449–466, 1999.

[40] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In *Financial Cryptography*, pages 190–207, 2003.

[41] Oded Goldreich. A simple protocol for signing contracts. In *CRYPTO*, pages 133–136, 1983.

[42] Oded Goldreich. *Foundations of Cryptography, Volume 2 Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[43] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *J. Cryptology*, 9(3):167–190, 1996.

[44] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. *J. Cryptology*, 19(3):241–340, 2006.

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

[46] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[47] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO*, pages 77–93, 1990.

[48] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[49] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In *STOC*, pages 413–422, 2008.

[50] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In *TCC*, pages 91–108, 2010.

[51] S. Dov Gordon and Jonathan Katz. Rational secret sharing, revisited. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 229–241. Springer, 2006.

[52] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *EUROCRYPT*, pages 157–176, 2010.

[53] Joseph Y. Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: extended abstract. In *STOC*, pages 623–632, 2004.

[54] J Håstad and A Shamir. The cryptographic security of truncated linearly related variables. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 356–362, New York, NY, USA, 1985. ACM.

[55] Russell Impagliazzo and Moti Yung. Direct minimum-knowledge computations. In *CRYPTO*, pages 40–51, 1987.

[56] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[57] Sergei Izmalkov, Silvio Micali, and Matt Lepinski. Rational secure computation and ideal mechanism design. In *FOCS*, pages 585–595, 2005.

[58] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *STOC*, pages 11–20, 2007.

[59] Jonathan Katz. Bridging game theory and cryptography: Recent results and future directions. In *TCC*, pages 251–272, 2008.

[60] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[61] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[62] Joe Kilian, Eyal Kushilevitz, Silvio Micali, and Rafail Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 29(4):1189–1208, 2000.

[63] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In *TCC*, pages 320–339, 2008.

[64] Gillat Kol and Moni Naor. Games for exchanging information. In *STOC*, pages 423–432, 2008.

[65] Alptekin Küpçü and Anna Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS*, pages 488–507, 2010.

[66] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In *CT-RSA*, pages 252–267, 2010.

[67] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Complete fair SFE and coalition-safe cheap talk. In *23rd ACM Symposium on Principles of Distributed Computing (PODC)*, 2004.

[68] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptology*, 16(3):143–184, 2003.

[69] Michael Luby, Silvio Micali, and Charles Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *FOCS*, pages 11–21, 1983.

[70] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *CRYPTO*, pages 180–197, 2006.

[71] Ralph Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[72] Silvio Micali. Simultaneous electronic transactions. U.S. patent no. 5,666,420, continuation of application no. 408,551, filed March 21, 1995, issued September 9, 1997.

[73] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991.

[74] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *TCC*, pages 1–18, 2009.

[75] Shien Jin Ong, David C. Parkes, Alon Rosen, and Salil P. Vadhan. Fairness with an honest minority and a rational majority. In *TCC*, pages 36–53, 2009.

[76] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Optimal efficiency of optimistic contract signing. In *PODC*, pages 113–122, 1998.

[77] Benny Pinkas. Fair secure two-party computation. In *EUROCRYPT*, pages 87–105, 2003.

[78] Michael O. Rabin. Digitalized signatures. In *Foundations of Secure Computations*, pages 155–168, 1978.

[79] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981. http://eprint.iacr.org/2005/187.

[80] Michael O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983.

[81] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989.

[82] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[83] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[84] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[85] Tom Tedrick. How to exchange half a bit. In *CRYPTO*, pages 147–151, 1983.

[86] Tom Tedrick. Fair exchange of secrets. In *CRYPTO*, pages 434–438, 1984.

[87] Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *FOCS*, pages 23–30. IEEE, 1983.

[88] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

[89] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.